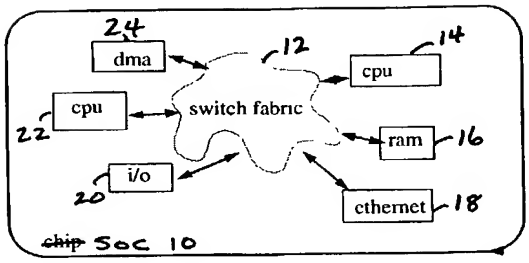
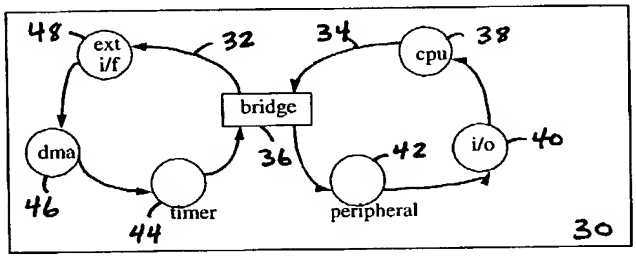
Fig. 1



(Prior Art)

Fig. 2



How Rings assign address space

Step1: align incoming address to self (to some power of 2)

Step2: assign the result to self address

Step3. next_addr = self_addr + self_addr_space; // number of register used locally

Step4: send down next_addr

Example:

Dma needs 16 addrs

Uart needs 4

Timer needs 256

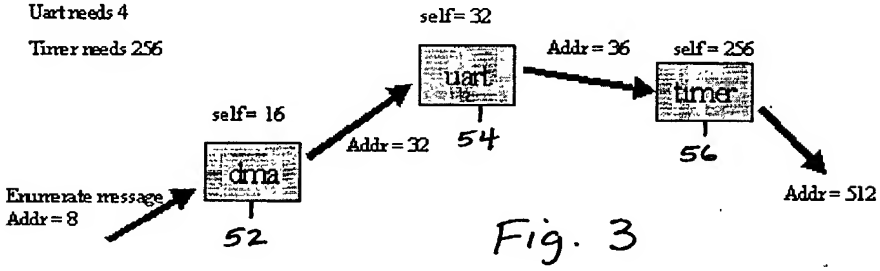
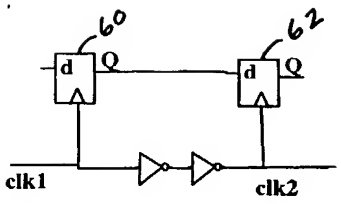


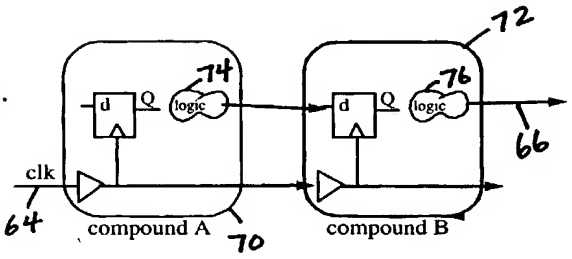
Fig. 3

Fig. 4



If the delay between "clk1" and "clk2" greater then the delay from Q to d of second flipflop, we have a race on our meaning right hand flipflop will sample the data of Q a whole clock period early.

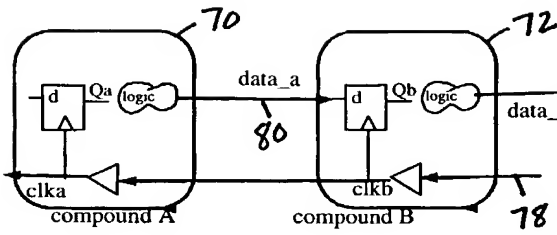
Fig. 5



clock runs with data

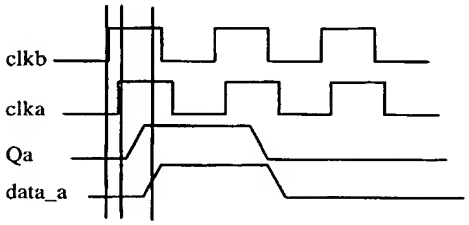
the problem is possible race. However, we control the logic on each flipflop leaving the compound, because it is always the same standart ring-interface module. we can ensure, that the delay will be at least enough. And more importantly easily checked after layout.

Fig. 6



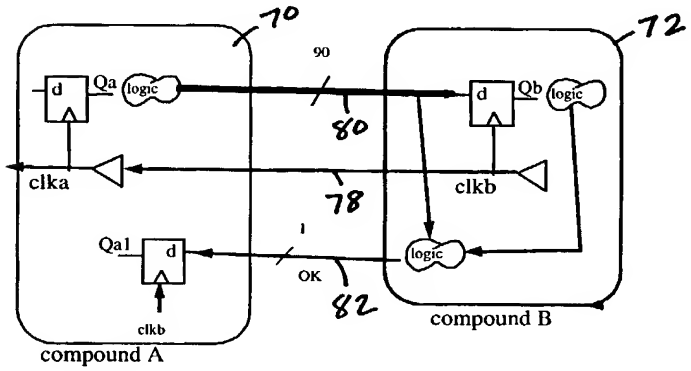
clock opposes data

this arrangement has the advantage of auto ensuring the no race condition (at least in this simple case) exists



data_a which changes after clkb, which is later then clka, is sampled by clka. NO RACE.

Fig. 7



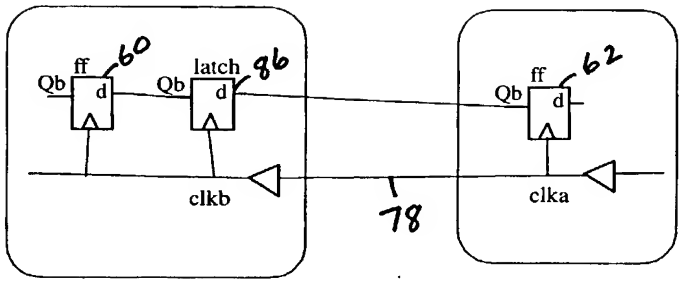


Fig. 8

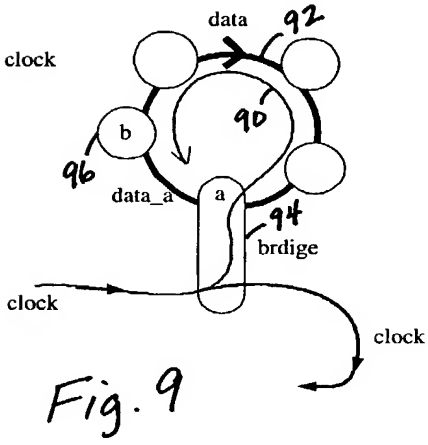
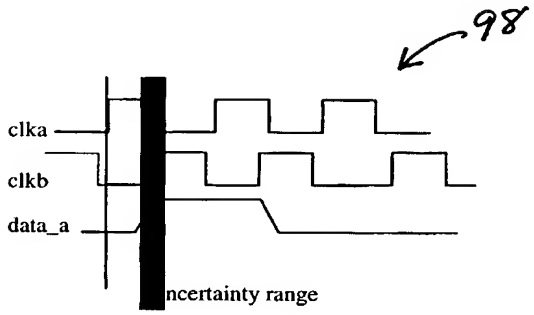


Fig. 9



data_a leaving the bridge goes to member "b" and there should be sampled by rising of clkb. clkb lags a lot behind clka of the bridge. As clearly seen from the waveforms, race is eminent. Here we should add latches for all the data lines (~90). Adding latch works however if the delay between clka and clkb is less then 75% of cycle time. otherwise the uncertainty kills the usable time. It sets hard limit on the number of ring members. Also keep in mind that latches needed on each OK signal between members of the ring

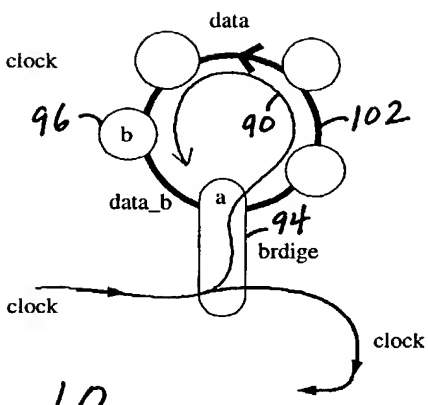
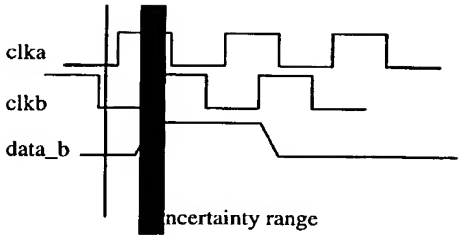
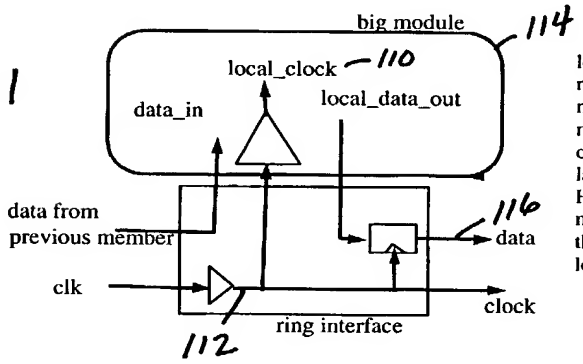


Fig. 10



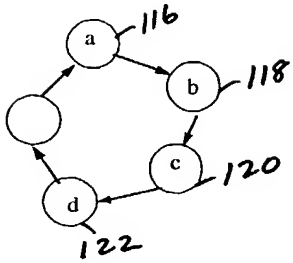
Here, data_b leaves member "b" to be sampled by clka in the bridge. But now clkb lags a lot behind clka. This actually works to our advantage, If the lag is smaller then better part of clock cycle. This solution looks better, because between adjacent members, we can take care to delay the datas beyond danger zone of clock delay, the OK signals are covered automatically, and last leg data is also covered. The only signal not safe is the OK from bridge to "b" member. It will need a latch in "b".

Fig. 11



local clock lags behind ring_interface clock of this module, because we presume the module is big. for data_coming out, it is not a problem, it changes later then ring-i/f flipflops clock. However for data entering the module from previous member, the race is a possibility we must look into.

Fig. 12



if module "a" sends a message to module "b", ring works fine. However if most of the traffic is from "c" to "b", this is more expensive in terms of latency.

Another problem is "peak latency". Suppose that , "a" transmits mostly to "d" and "b" mostly to "c" In this case communication between "b" and "c" suffers degradation in case that peak traffic coincide.

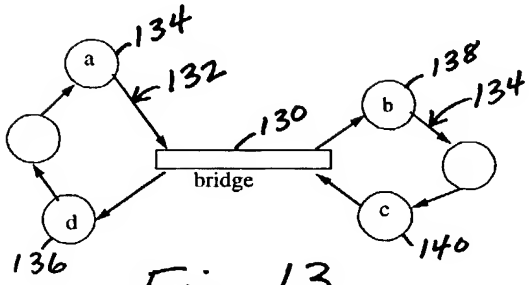
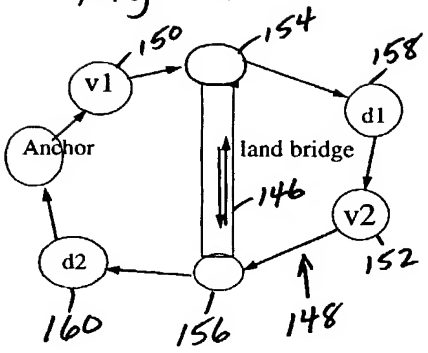


Fig. 13

Fig. 14



Land bridge gets its name from the fact that it is a luxury. It spans across connected modules. The idea is simple. When V2 sends message to D1 it gets to one side of the bridge. This side analyzes the destination address and by some magic (explained later) decides to short-cut the path. The message re-appears at the other end of the bridge and gets fast to D1. By same magic, message fromV1 to D2 get bypassed also. message fromV1 to D1 is treated directly.

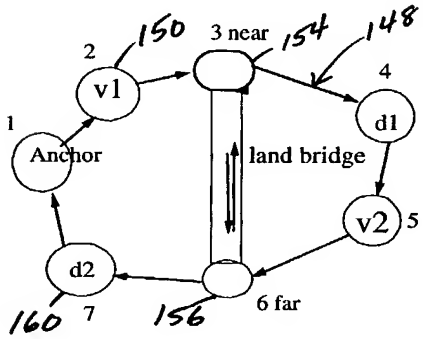
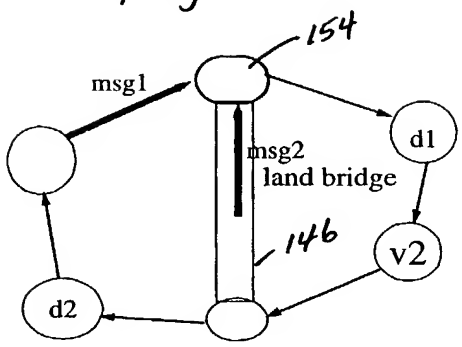


Fig. 15

Enumeration is started by "Anchor" which assigns address=1 to itself. results of enumeration are labels 1 to 7. land bridge gets two addresses , as if it were not one module. there is "near" end, that got enumeration label "3", and the "far" end marked 6.

Fig. 16



msg1 and msg2 arrive at the same time. the bridge end must make a decision which message to forward first.

It can be shown that unwise decision can lead to freezout, deadlock and option price dropping to 5$.

Therefore MSG2 gets the priority.

Fig. 17

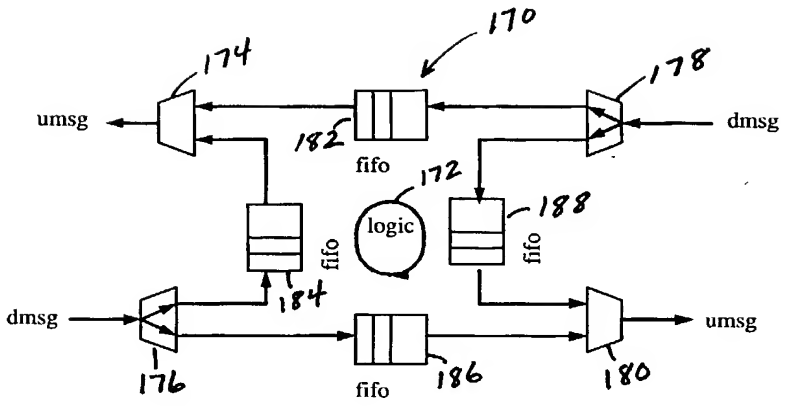
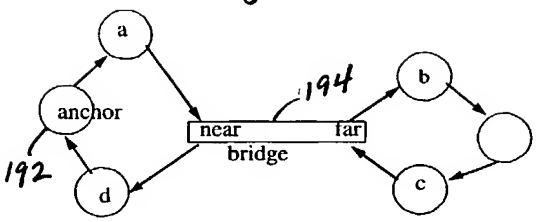

170
174
178
umsg
dmsg
182
fifo
172
logic
188
fifo
fifo
184
dmsg
umsg
176
fifo
186
180

Fig. 18



Bridge takes responcibility for strays, but only at the "far" end. During enumeration, bridge is "polarized" to have near and far end. Near is the end first struck by enumeration message.

So we have exactly one enforcer for each ring.

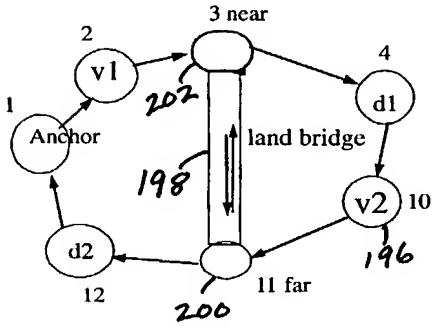


Fig. 19

In land bridge ring, the situation is trickier. If V2 send message to address==5. The land bridge divert at 11/far end. it will re-appear at 3 and start cycling forever.

We have to define an algorithm that will take care of all cases.

Luckily there is a way.

Land Bridge deals only with messages arriving at the far end and being diverted. It marks and monitors only those. Messages arriving at near end, keep their markings. Messages at fdar end going through, are left alone.

Fig. 20

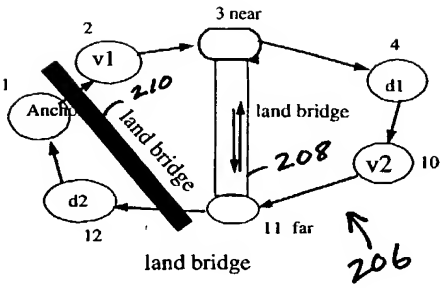
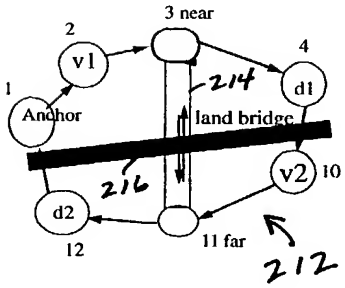


Fig. 21



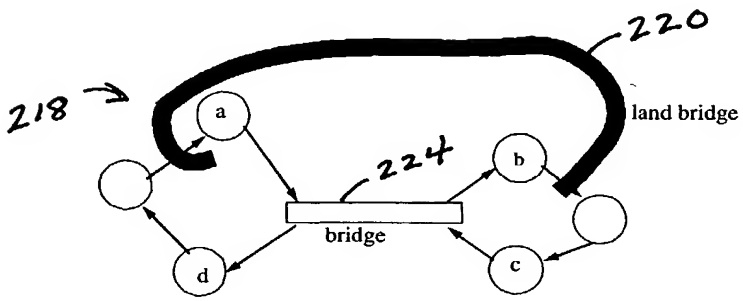


Fig. 22

Fig. 23

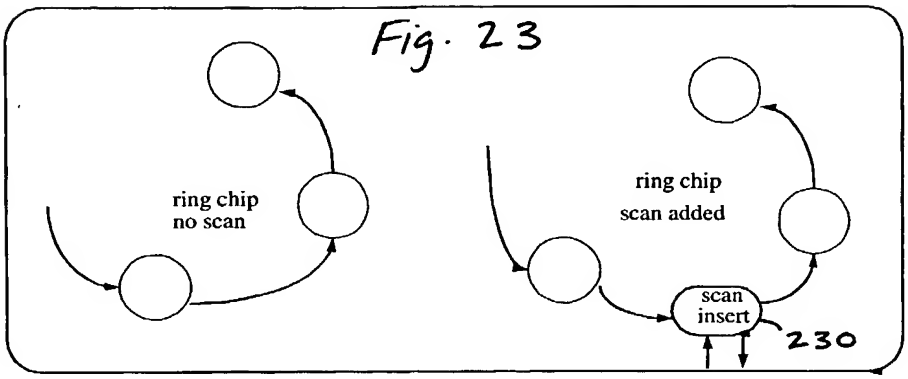


Fig. 24

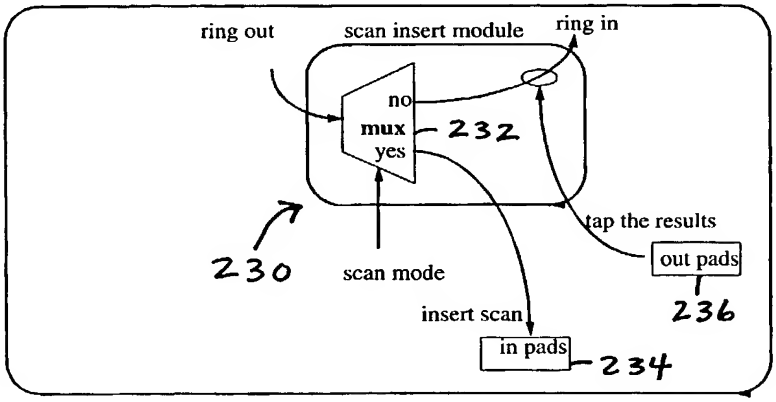
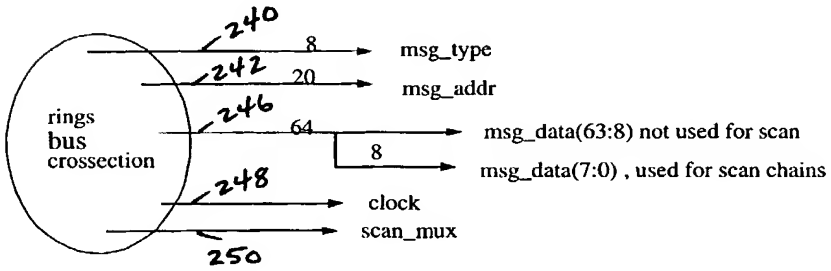


Fig. 25



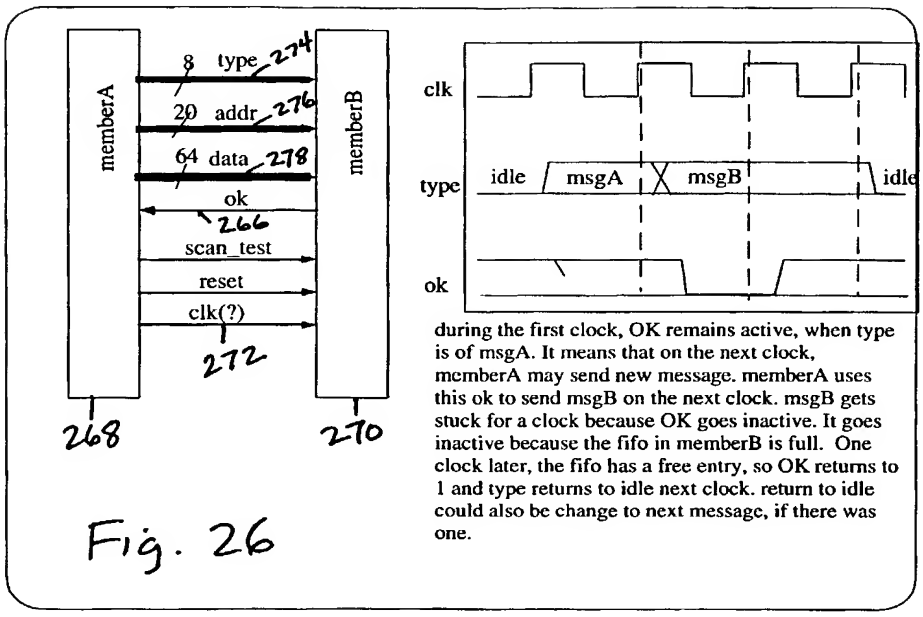

memberA
memberB
8 type 274
20 addr 276
64 data 278
ok
266
scan_test
reset
clk(?)
272
268
270
clk
type
idle
msgA
msgB
idle
ok
during the first clock, OK remains active, when type is of msgA. It means that on the next clock, memberA may send new message. memberA uses this ok to send msgB on the next clock. msgB gets stuck for a clock because OK goes inactive. It goes inactive because the fifo in memberB is full. One clock later, the fifo has a free entry, so OK returns to 1 and type returns to idle next clock. return to idle could also be change to next message, if there was one.

Fig. 26

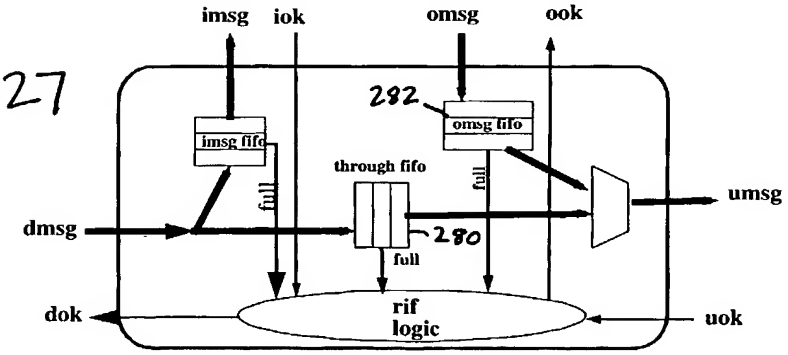
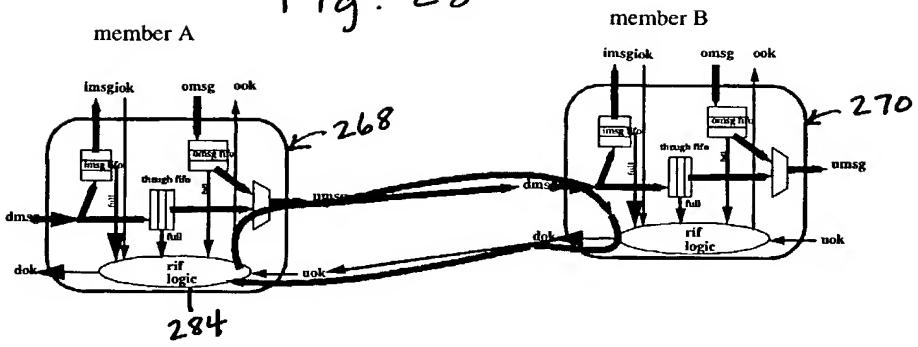

imsg
iok
omsg
ook
282
omsg fifo
imsg fifo
full
through fifo
full
umsg
dmsg
280
full
dok
rif
logic
uok

Fig. 27

Fig. 28



The incoming messages are examined first to see if it is supervisor or work/program. Work/program messages have address field. We check if it is our address. Since we know that our address is aligned to our power of 2, The address mask (named split mask) causes only certain number if upper bits to be compared. The lower part of the address is passed inside as internal address. The upper bits are compared against self-address register. This register gets its value during enumeration protocol. The lower part of this register is always masked,. Hopefully synthesis will delete the unused bits implementation.

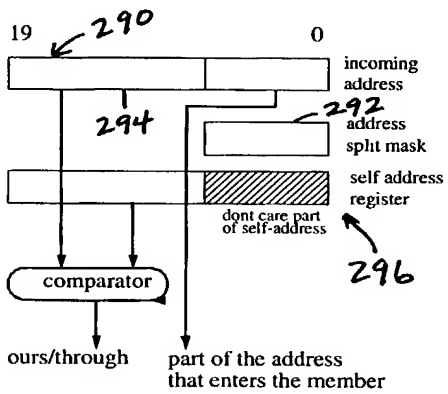


Fig. 29

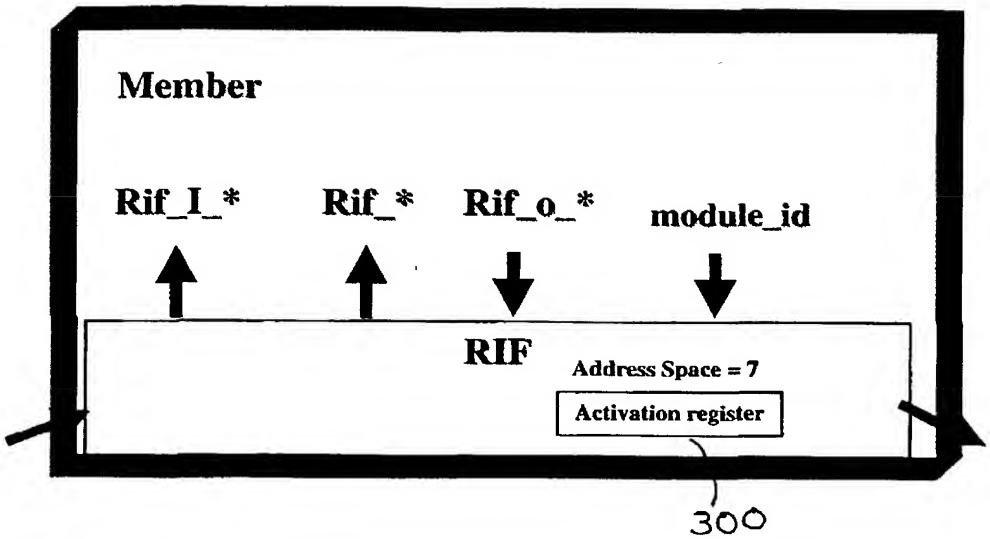

Member
Rif_I_*
Rif_*
Rif_o_*
module_id
RIF
Address Space = 7
Activation register
300

Fig. 30

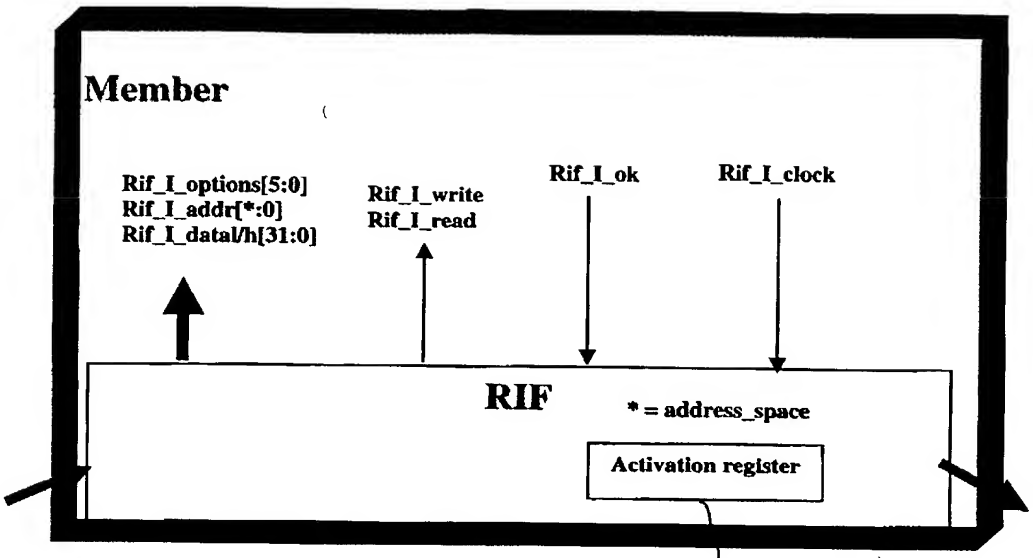

Member
Rif_I_options[5:0]
Rif_I_addr[*:0]
Rif_I_data[/h[31:0]
Rif_I_write
Rif_I_read
Rif_I_ok
Rif_I_clock
RIF
* = address_space
Activation register
300

Fig. 31

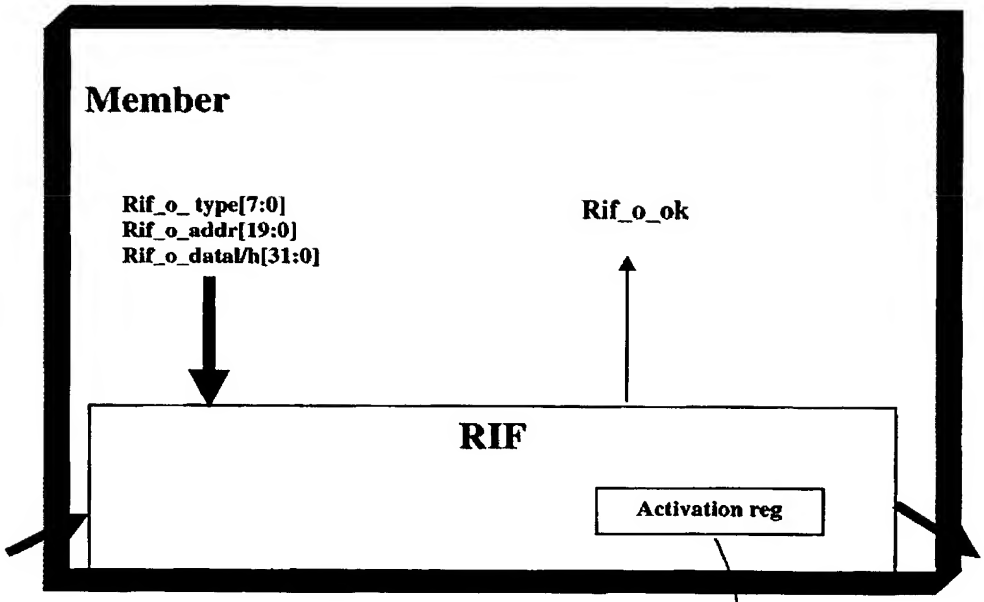

Member
Rif_o_ type[7:0]
Rif_o_addr[19:0]
Rif_o_datal/h[31:0]
Rif_o_ok
RIF
Activation reg
300

Fig. 32

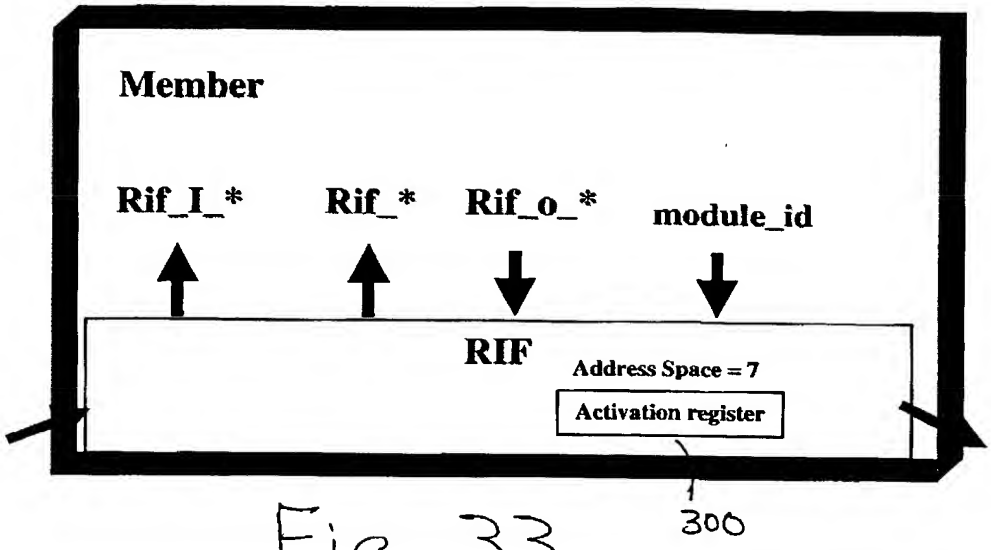
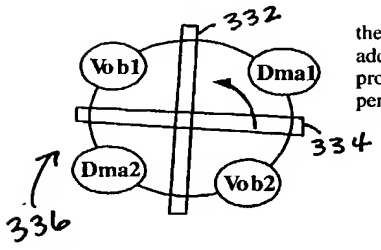

Member
Rif_I_*
Rif_*
Rif_o_*
module_id
RIF
Address Space = 7
Activation register
300

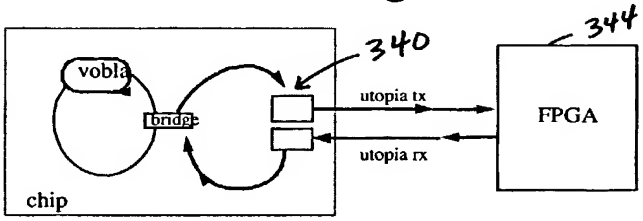
Fig. 33

Fig. 34



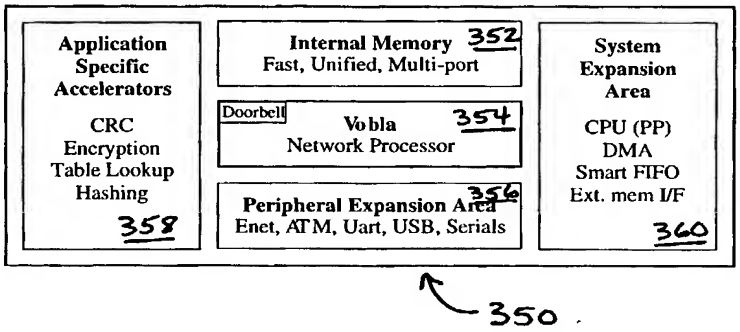
the second land bridge solves most traffic problems, but adds 4 clocks in the overall ring length. This is not a big problem becausc no message should travel the whole perimiter.

Fig. 35



The utopia interface is forced into mode that communicates in messages, not cells. We using the I/O and maybe some of the logic.

Fig. 36



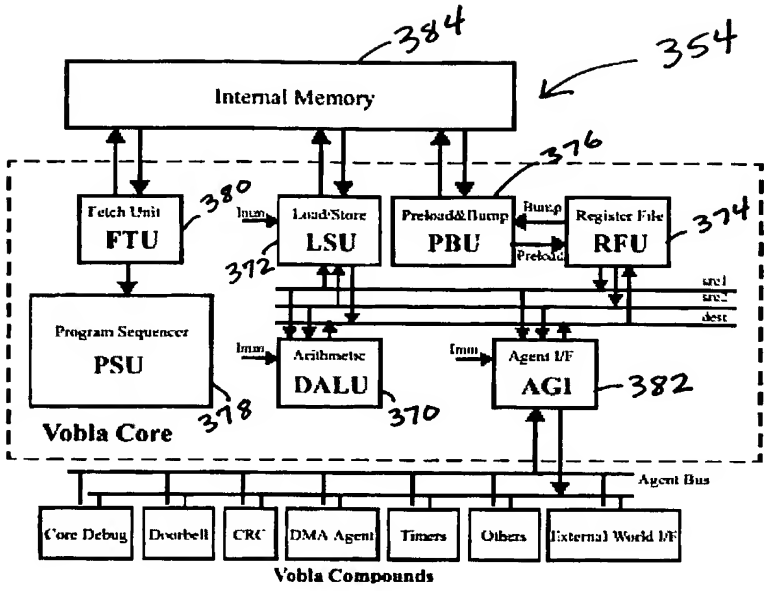
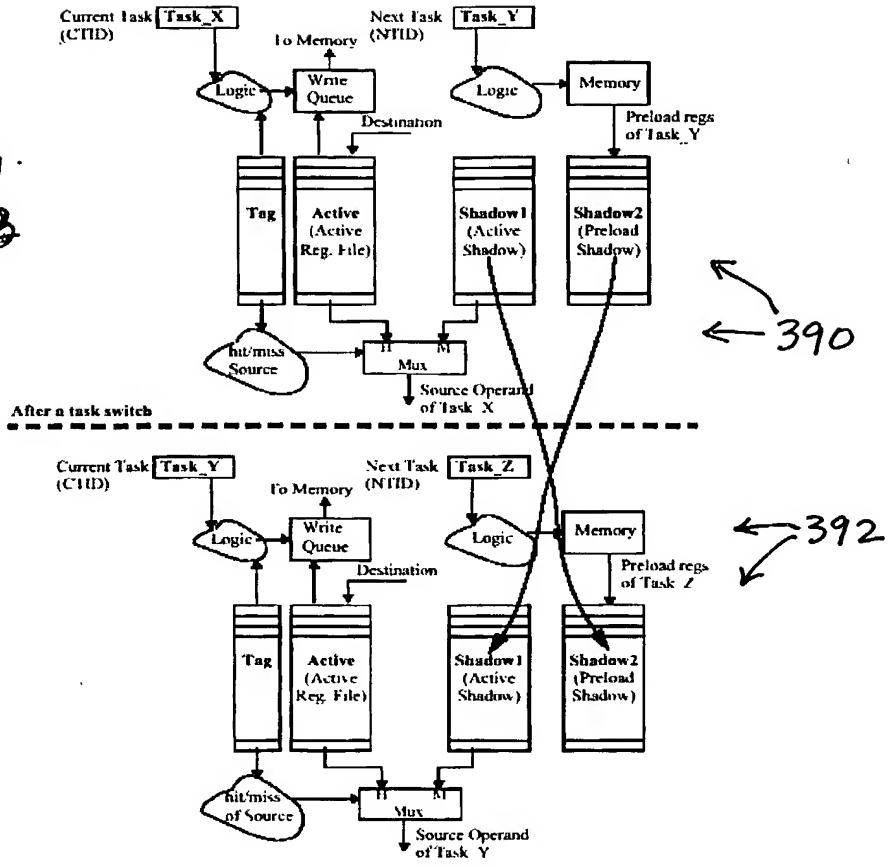

384
354
Internal Memory
376
Fetch Unit
FTU
380
Imm
Load/Store
LSU
372
Preload&Bump
PBU
Bump
Preload
Register File
RFU
374
src1
src2
dest
Program Sequencer
PSU
Imm
Arithmetic
DALU
370
Imm
Agent I/F
AGI
382
378
Vobla Core
Agent Bus
Core Debug
Doorbell
CRC
DMA Agent
Timers
Others
External World I/F
Vobla Compounds

Fig. 37

Fig. 3B


Current Task (CTID) Task_X
To Memory
Next Task (NTID) Task_Y
Logic
Write Queue
Logic
Memory
Destination
Preload regs of Task_Y
Tag
Active (Active Reg. File)
Shadow1 (Active Shadow)
Shadow2 (Preload Shadow)
390
hit/miss Source
H M
Mux
Source Operand of Task_X
After a task switch
Current Task (CTID) Task_Y
To Memory
Next Task (NTID) Task_Z
Logic
Write Queue
Logic
Memory
392
Destination
Preload regs of Task_Z
Tag
Active (Active Reg. File)
Shadow1 (Active Shadow)
Shadow2 (Preload Shadow)
hit/miss of Source
H M
Mux
Source Operand of Task_Y

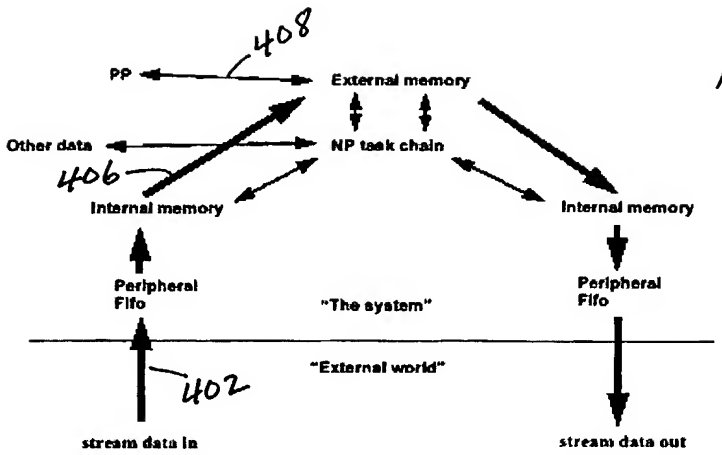

408
PP
External memory
Other data
NP task chain
406
Internal memory
Internal memory
Peripheral Fifo
"The system"
Peripheral Fifo
"External world"
402
stream data in
stream data out

Fig. 39

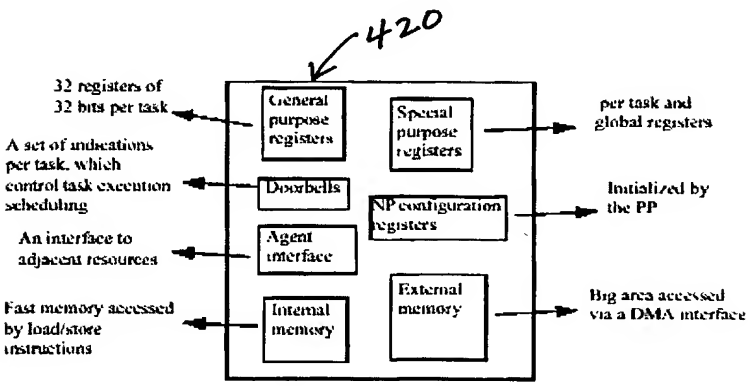

420
32 registers of 32 bits per task
A set of indications per task, which control task execution scheduling
An interface to adjacent resources
Fast memory accessed by load/store instructions
General purpose registers
Doorbells
Agent interface
Internal memory
Special purpose registers
NP configuration registers
External memory
per task and global registers
Initialized by the PP
Big area accessed via a DMA interface

Fig. 40

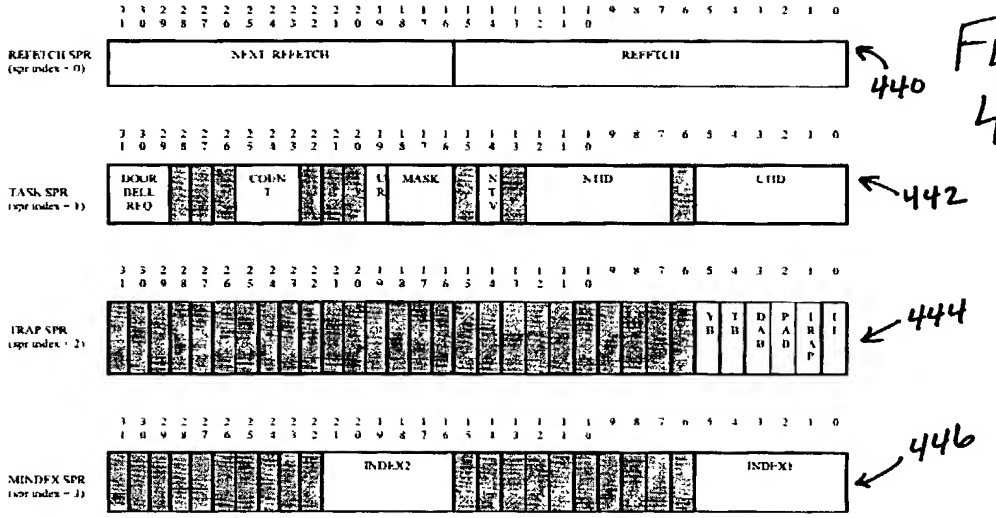
Fig. 41

R1 register:

s - sticky bit
eq - equal/zero
lt - less then/negative
gt - greater then/positive
c - carry
mb - reflection of the RAM multi-reader busy indication.

430

Fig. 42



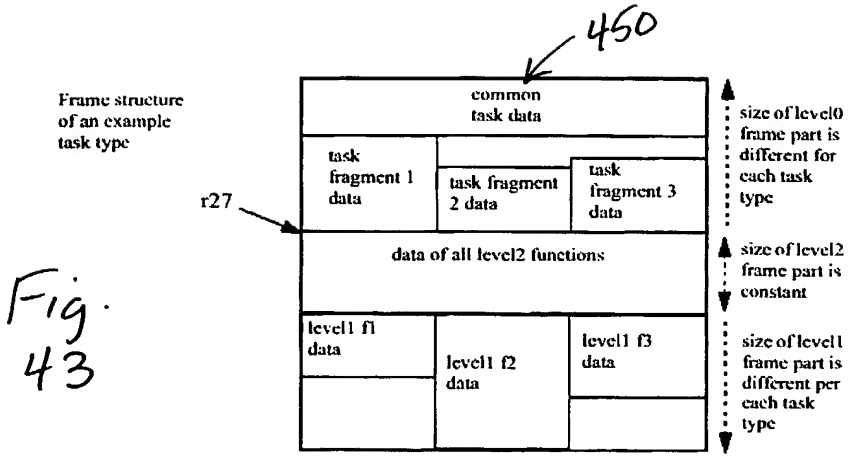

450
Frame structure of an example task type
common task data
task fragment 1 data
task fragment 2 data
task fragment 3 data
r27
data of all level2 functions
level1 f1 data
level1 f2 data
level1 f3 data
size of level0 frame part is different for each task type
size of level2 frame part is constant
size of level1 frame part is different per each task type

Fig. 43

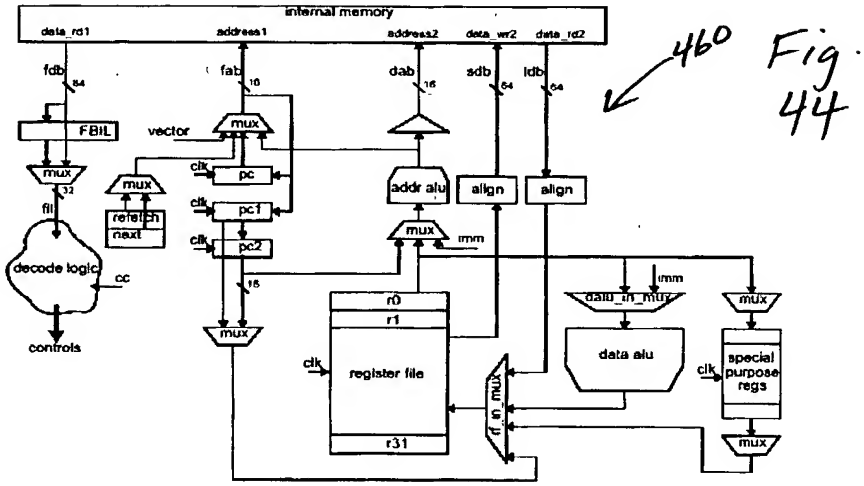

internal memory
data_rd1
address1
address2
data_wr2
data_rd2
fdb
fab
dab
sdb
ldb
460
FBIL
vector
mux
pc
pc1
pc2
refetch
next
fil
decode logic
cc
controls
addr alu
align
imm
r0
r1
register file
r31
data alu
special purpose regs
clk

Fig. 44

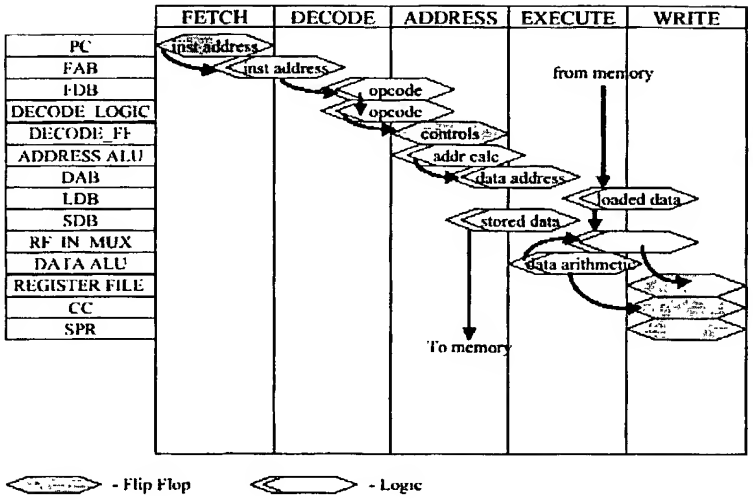

FETCH
DECODE
ADDRESS
EXECUTE
WRITE
PC
FAB
FDB
DECODE_LOGIC
DECODE_FF
ADDRESS ALU
DAB
LDB
SDB
RF_IN_MUX
DATA ALU
REGISTER FILE
CC
SPR
inst address
inst address
opcode
opcode
controls
addr calc
data address
from memory
loaded data
stored data
data arithmetic
To memory
- Flip Flop
- Logic
480

Fig. 45

Fig. 46

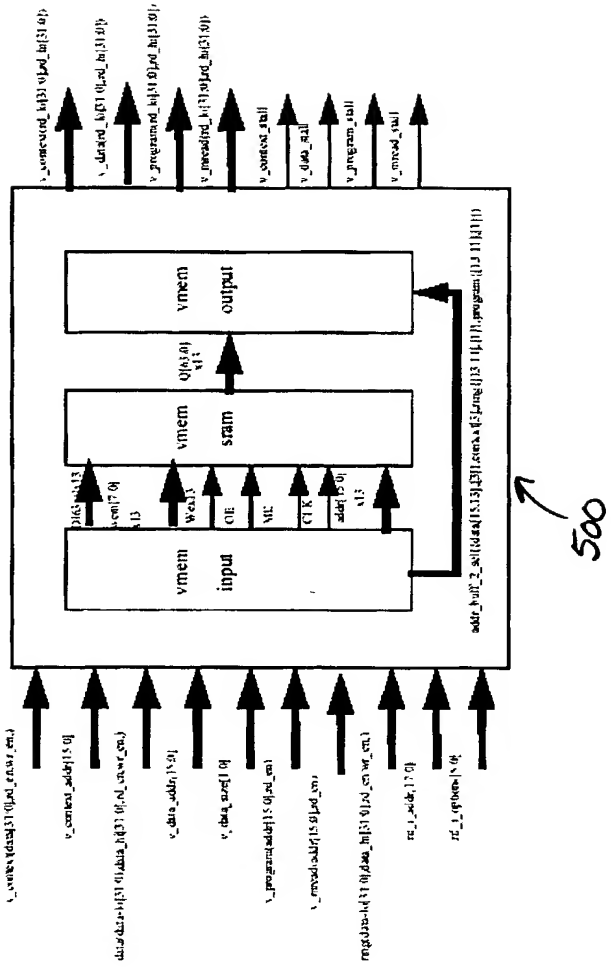
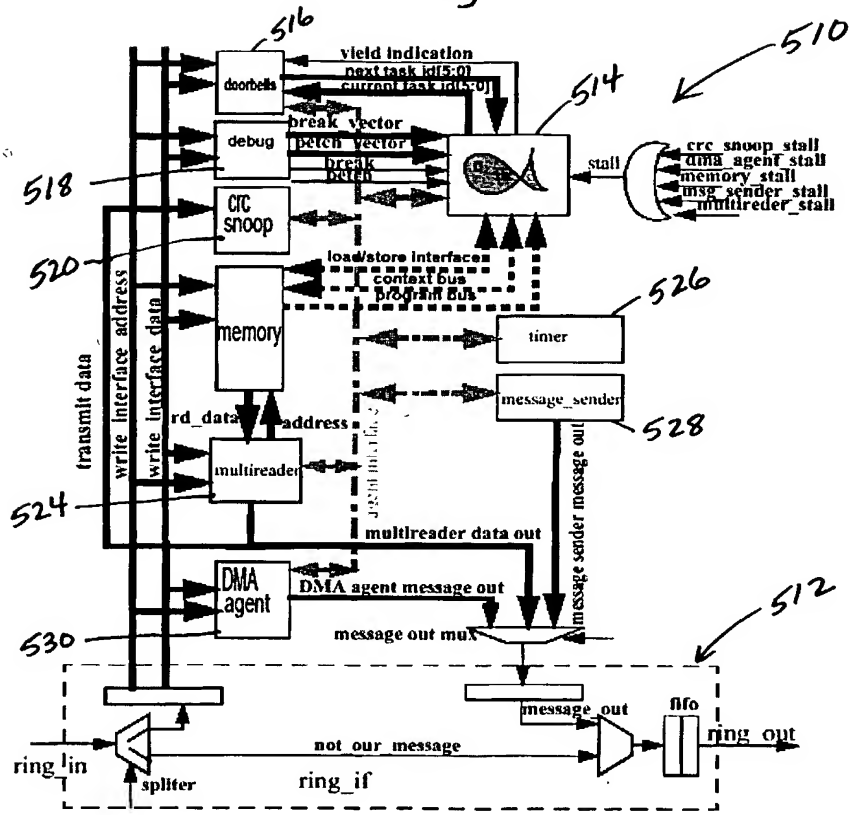


Fig. 47


516
yield indication
next task id[5:0]
current task id[5:0]
doorbells
510
514
break vector
debug
patch vector
break
patch
stall
crc_snoop_stall
dma_agent_stall
memory_stall
msg_sender_stall
multireder_stall
518
crc snoop
520
load/store interface
context bus
program bus
memory
526
timer
message_sender
528
transmit data
write interface address
write interface data
rd_data
address
multireader
524
multireader data out
message sender message out
DMA agent
DMA agent message out
512
530
message out mux
message_out
fifo
ring_out
not_our_message
ring_in
splitter
ring_if

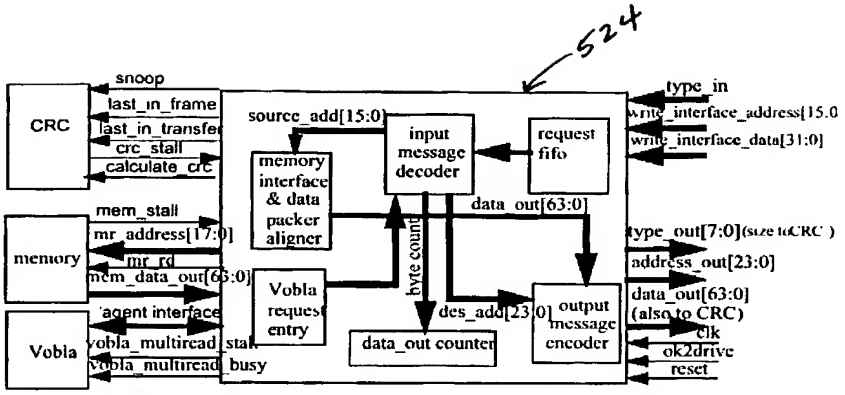


Fig. 48

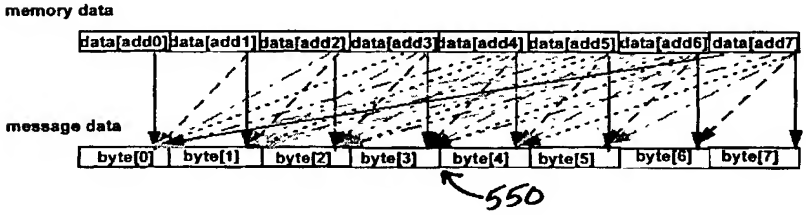


Fig. 49

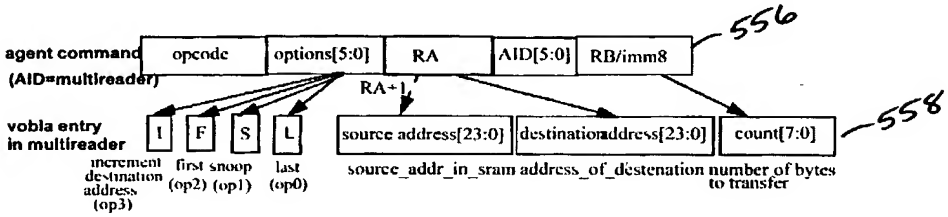
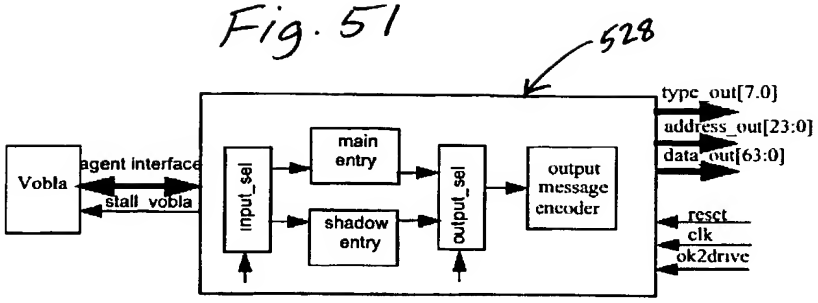


Fig. 50

Fig. 51



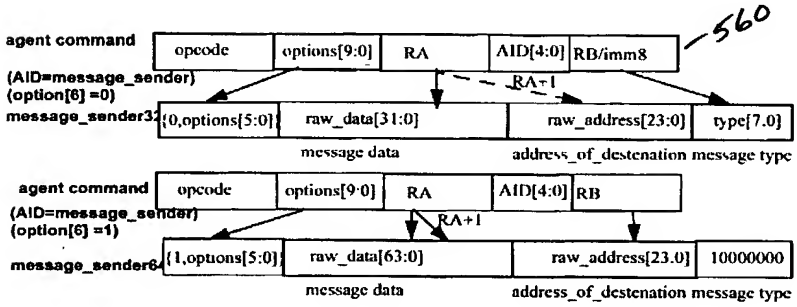


Fig. 52

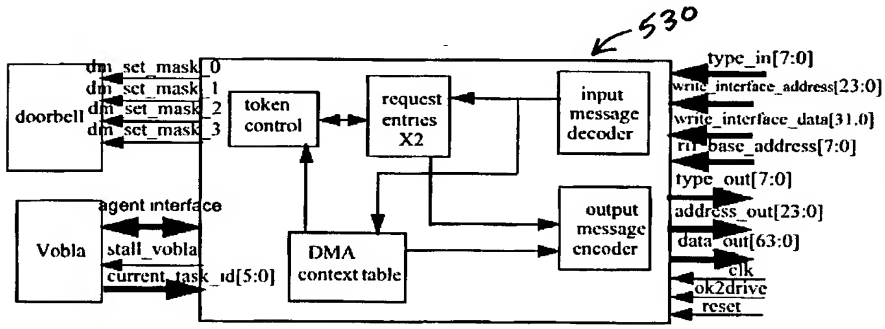
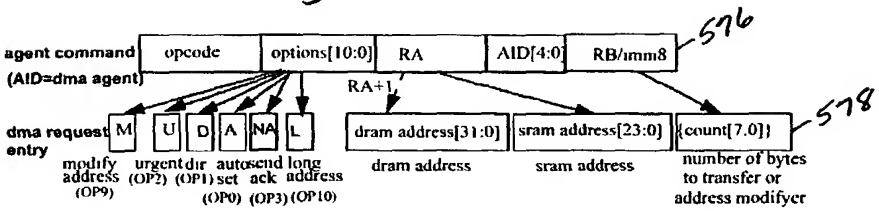


Fig. 53

Fig. 54



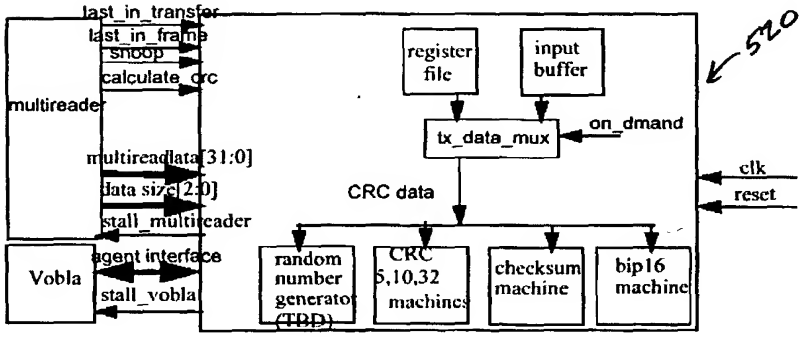
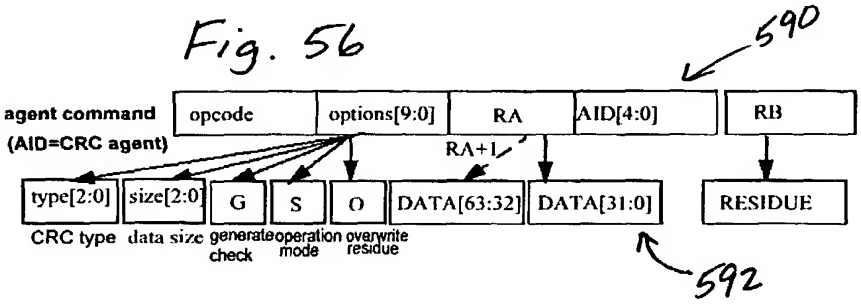


Fig. 55

Fig. 56



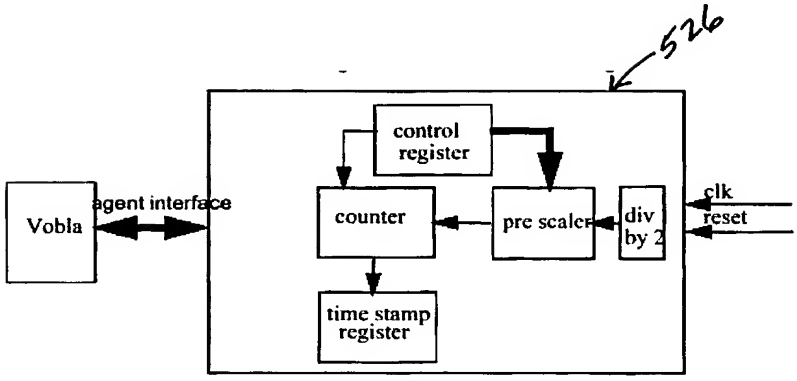

526
control register
Vobla
agent interface
counter
pre scaler
div by 2
clk
reset
time stamp register

Fig. 57

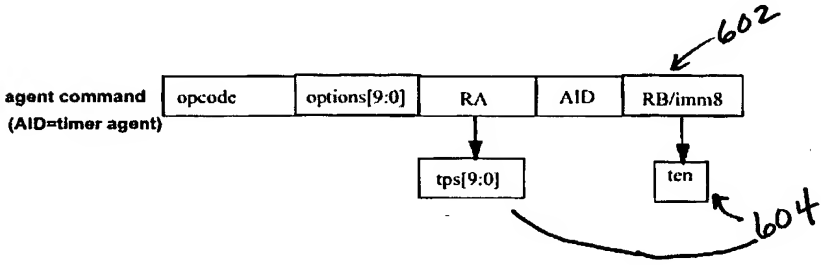

602
agent command
(AID=timer agent)
opcode
options[9:0]
RA
AID
RB/imm8
tps[9:0]
ten
604

Fig. 58

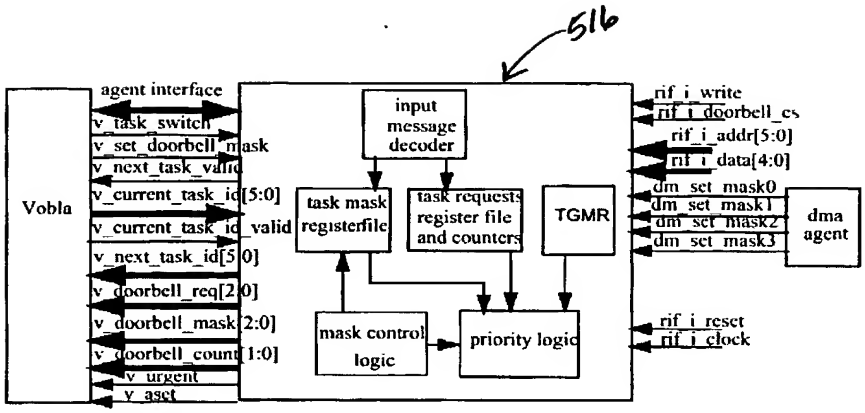

516
agent interface
Vobla
v_task_switch
v_set_doorbell_mask
v_next_task_valid
v_current_task_id[5:0]
v_current_task_id_valid
v_next_task_id[5:0]
v_doorbell_req[2:0]
v_doorbell_mask[2:0]
v_doorbell_count[1:0]
v_urgent
v_aset
input message decoder
task mask registerfile
task requests register file and counters
TGMR
mask control logic
priority logic
rif_i_write
rif_i_doorbell_cs
rif_i_addr[5:0]
rif_i_data[4:0]
dm_set_mask0
dm_set_mask1
dm_set_mask2
dm_set_mask3
dma agent
rif_i_reset
rif_i_clock

Fig. 59

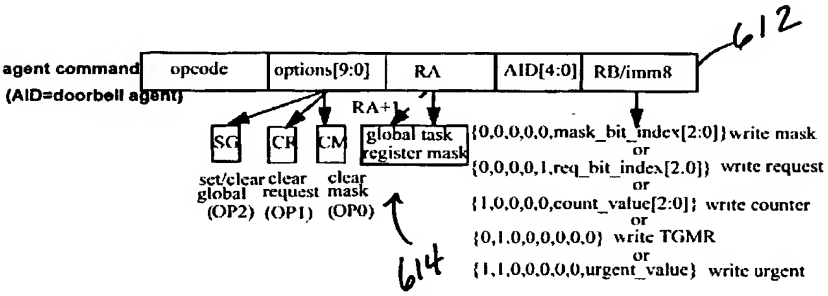

612
agent command
(AID=doorbell agent)
opcode
options[9:0]
RA
AID[4:0]
RB/imm8
RA+1
SG
CR
CM
global task register mask
set/clear global (OP2)
clear request (OP1)
clear mask (OP0)
614
{0,0,0,0,0,mask_bit_index[2:0]} write mask
or
{0,0,0,0,1,req_bit_index[2.0]} write request
or
{1,0,0,0,0,count_value[2:0]} write counter
or
{0,1,0,0,0,0,0,0} write TGMR
or
{1,1,0,0,0,0,0,urgent_value} write urgent

Fig. 60

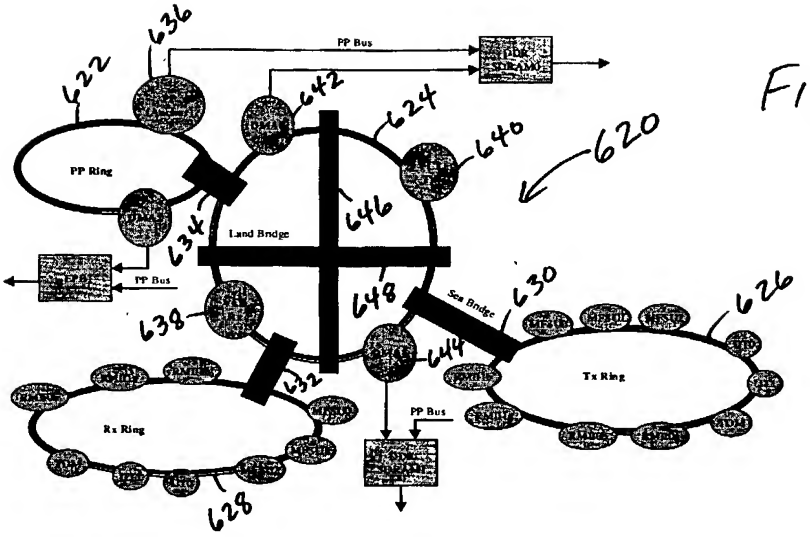

PP Bus
PP Ring
Land Bridge
PP Bus
Scu Bridge
Tx Ring
Rx Ring
PP Bus
620
622
624
626
628
630
632
634
636
638
640
642
644
646
648

Fig. 61

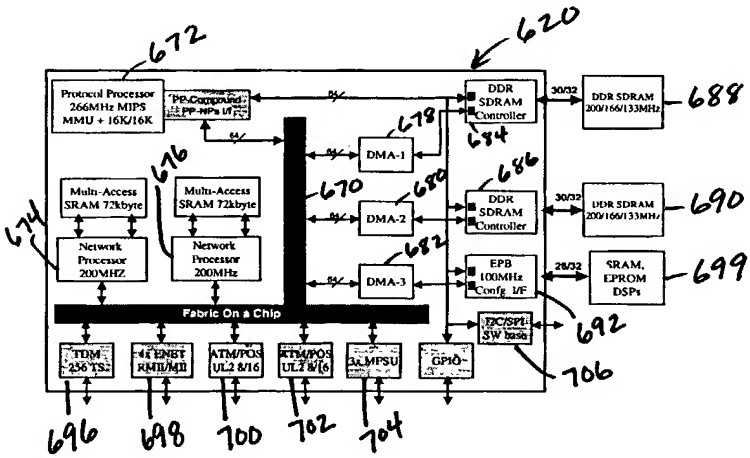

620
672
Protocol Processor 266MHz MIPS MMU + 16K/16K
PP-Coupling PP-HPs I/F
DDR SDRAM Controller
DDR SDRAM 200/166/133MHz
688
684
678
DMA-1
676
Multi-Access SRAM 72kbyte
Multi-Access SRAM 72kbyte
670
DMA-2
680
686
DDR SDRAM Controller
DDR SDRAM 200/166/133MHz
690
674
Network Processor 200MHz
Network Processor 200MHz
682
DMA-3
EPB 100MHz Config I/F
SRAM, EPROM DSPs
699
Fabric On a Chip
I2C/SPI SW basis
692
706
TDM 256 TS
4x ENET RMII/MII
ATM/POS UL2 8/16
ATM/POS UL2 8/16
3x MPSU
GPIO
696
698
700
702
704

Fig. 62

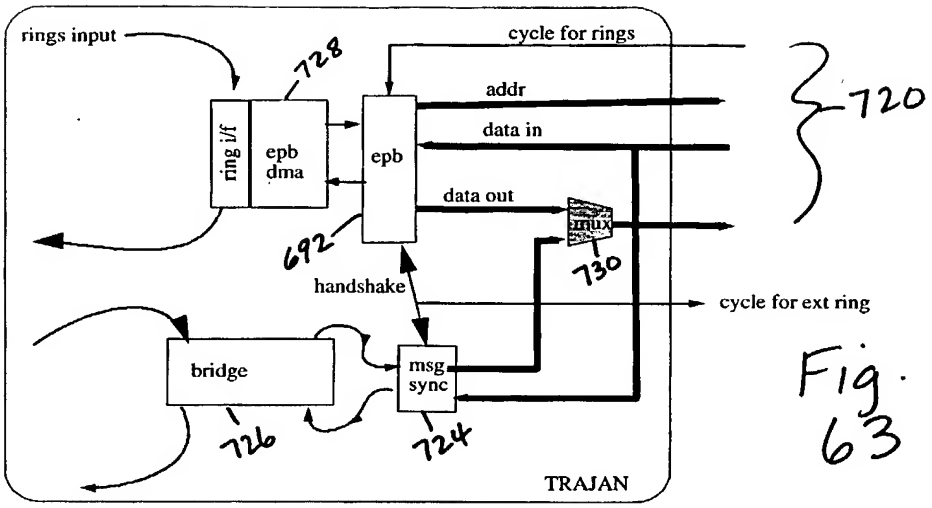

rings input
cycle for rings
728
addr
data in
ring i/f
epb dma
epb
720
data out
692
730
handshake
cycle for ext ring
bridge
msg sync
726
724
TRAJAN

Fig. 63

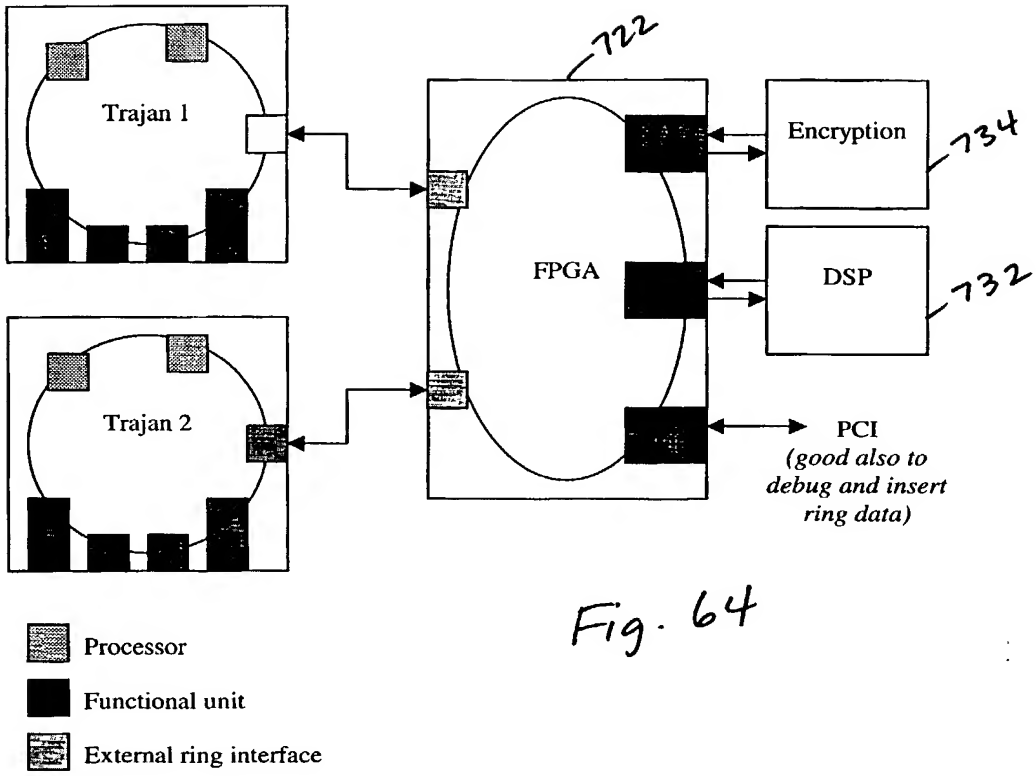
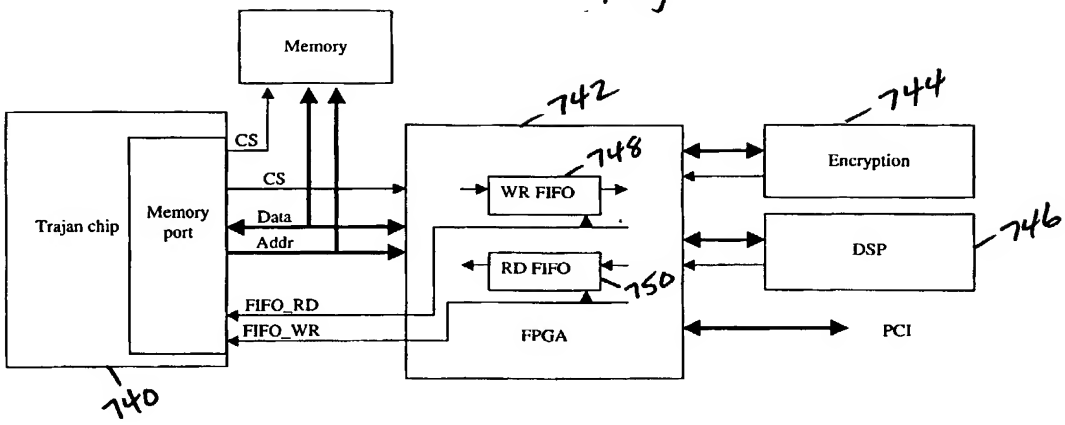

Trajan 1
Trajan 2
722
FPGA
Encryption
734
DSP
732
PCI
(good also to debug and insert ring data)
Processor
Functional unit
External ring interface

Fig. 64

Fig. 65


Memory
742
744
CS
748
CS
WR FIFO
Encryption
Trajan chip
Memory port
Data
Addr
RD FIFO
DSP
746
750
FIFO_RD
FIFO_WR
FPGA
PCI
740

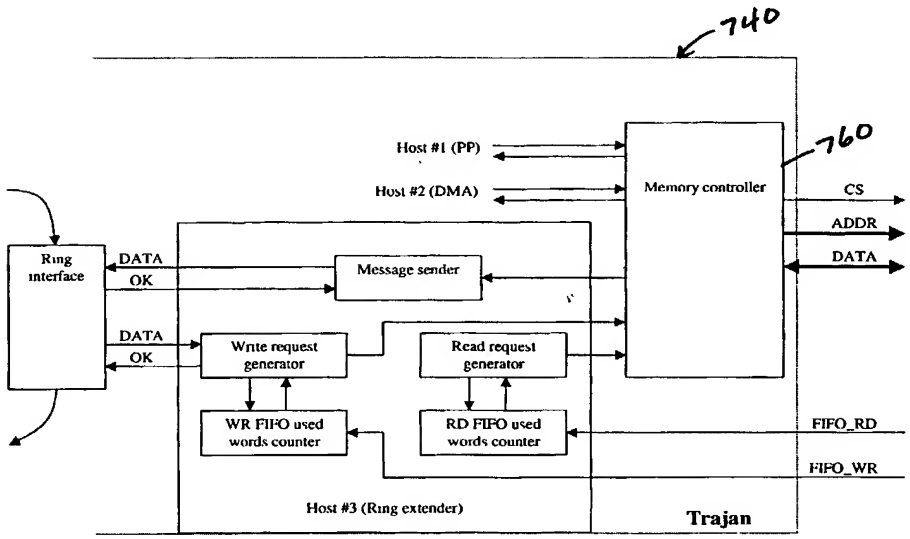

740
Host #1 (PP)
760
Host #2 (DMA)
Memory controller
CS
ADDR
DATA
Ring interface
DATA
OK
Message sender
DATA
OK
Write request generator
Read request generator
WR FIFO used words counter
RD FIFO used words counter
FIFO_RD
FIFO_WR
Host #3 (Ring extender)
Trajan

Fig. 66

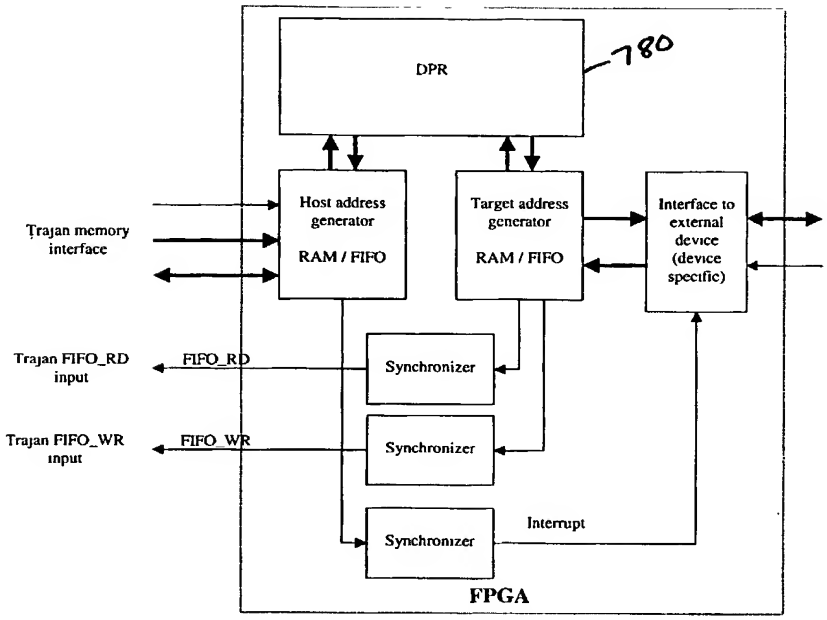

DPR
780
Host address generator
RAM / FIFO
Target address generator
RAM / FIFO
Interface to external device (device specific)
Trajan memory interface
Trajan FIFO_RD input
FIFO_RD
Synchronizer
Trajan FIFO_WR input
FIFO_WR
Synchronizer
Synchronizer
Interrupt
FPGA

Fig. 67

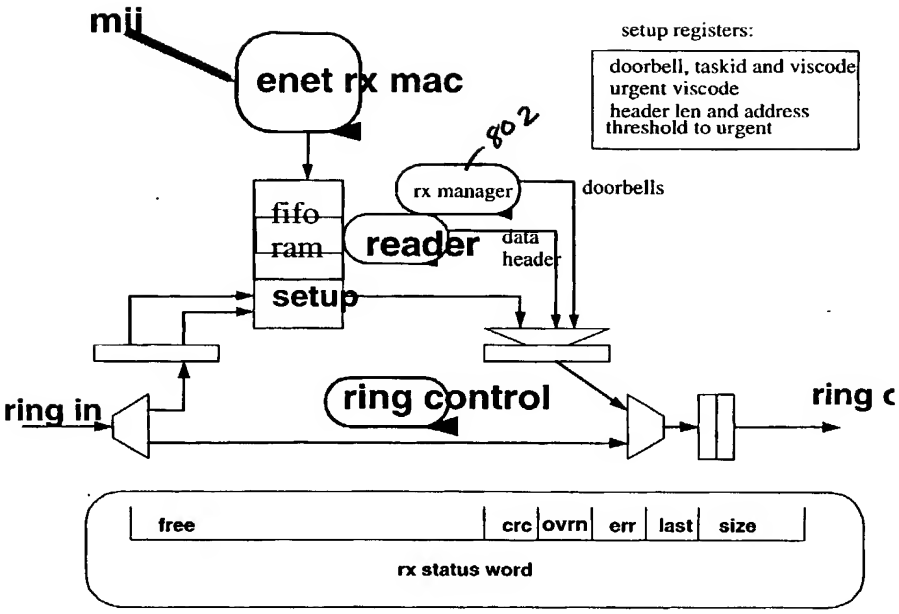

mii
enet rx mac
setup registers:
doorbell, taskid and viscode
urgent viscode
header len and address
threshold to urgent
602
rx manager
doorbells
fifo
ram
reader
data
header
setup
ring in
ring control
ring c
free
crc
ovrn
err
last
size
rx status word

Fig. 68

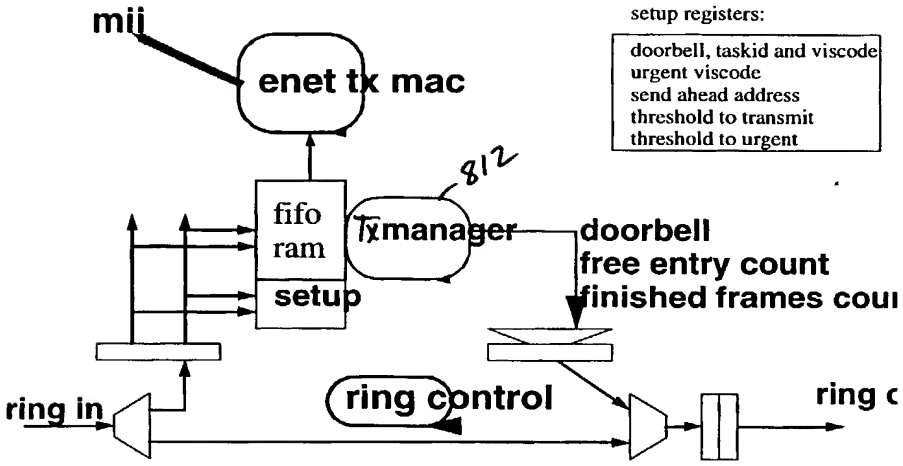

mii
enet tx mac
setup registers:
doorbell, taskid and viscode
urgent viscode
send ahead address
threshold to transmit
threshold to urgent
812
fifo ram
setup
Tx manager
doorbell
free entry count
finished frames cou
ring in
ring control
ring c

Fig. 69

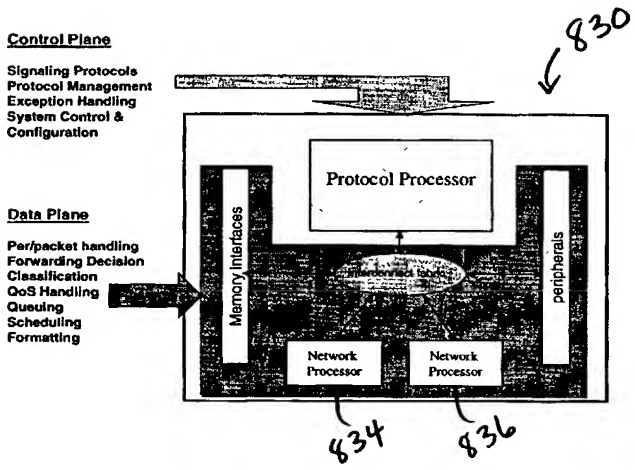

Control Plane
Signaling Protocols
Protocol Management
Exception Handling
System Control &
Configuration
Data Plane
Per/packet handling
Forwarding Decision
Classification
QoS Handling
Queuing
Scheduling
Formatting
830
Protocol Processor
Memory Interfaces
peripherals
Network Processor
Network Processor
834
836

Fig. 70

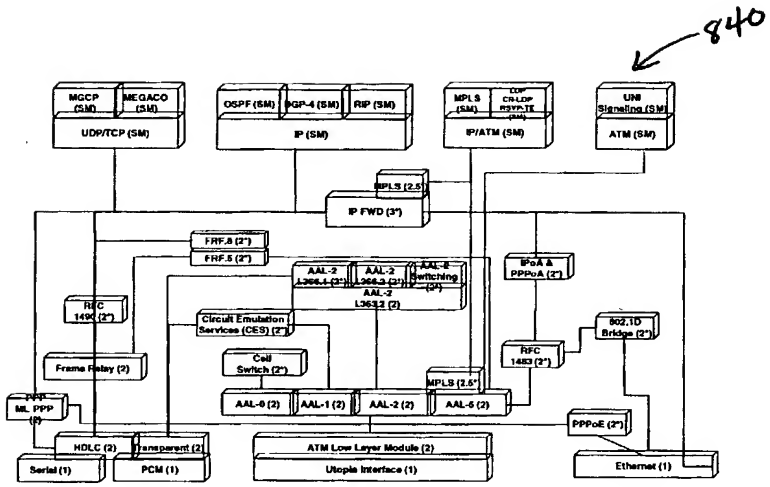

840
MGCP (SM)
MEGACO (SM)
UDP/TCP (SM)
OSPF (SM)
BGP-4 (SM)
RIP (SM)
IP (SM)
MPLS (SM)
IP/ATM (SM)
UNI Signaling (SM)
ATM (SM)
MPLS (2.5)
IP FWD (3*)
FRF.8 (2*)
FRF.5 (2*)
AAL-2
AAL-5
Switching (3*)
AAL-2 I.363.2 (2)
IPoA & PPPoA (2*)
RFC 1490 (2*)
Circuit Emulation Services (CES) (2*)
802.1D Bridge (2*)
Frame Relay (2)
Cell Switch (2*)
RFC 1483 (2*)
MPLS (2.5*)
AAL-0 (2)
AAL-1 (2)
AAL-2 (2)
AAL-5 (2)
ML PPP
PPPoE (2*)
HDLC (2)
Transparent (2)
Serial (1)
PCM (1)
ATM Low Layer Module (2)
Utopia Interface (1)
Ethernet (1)

Fig. 71

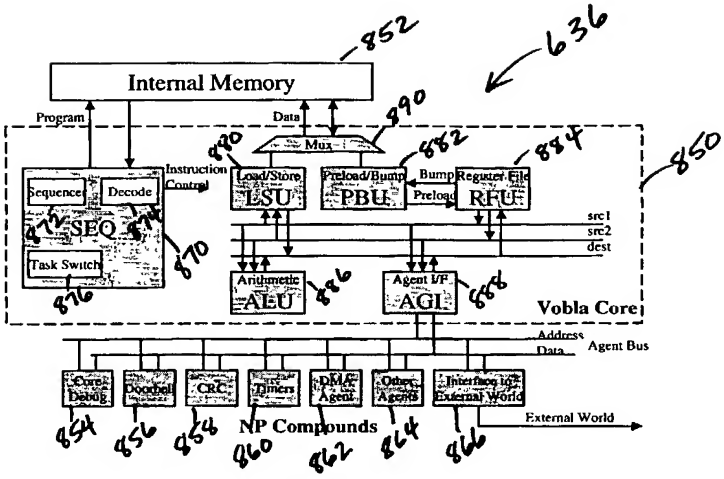

852
636
Internal Memory
Program
Data
890
Mux
880
882
884
850
Instruction Control
Sequence
Decode
SEQ
Task Switch
870
876
Load/Store
LSU
Preload/Bump
PBU
Bump
Preload
Register File
RFU
src1
src2
dest
Arithmetic
ALU
886
Agent I/F
AGL
888
Vobla Core
Address
Data
Agent Bus
Core Debug
Doorbell
CRC
Timers
DMC Agent
Other Agents
Interface to External World
External World
854
856
858
NP Compounds
860
862
864
866

Fig. 72

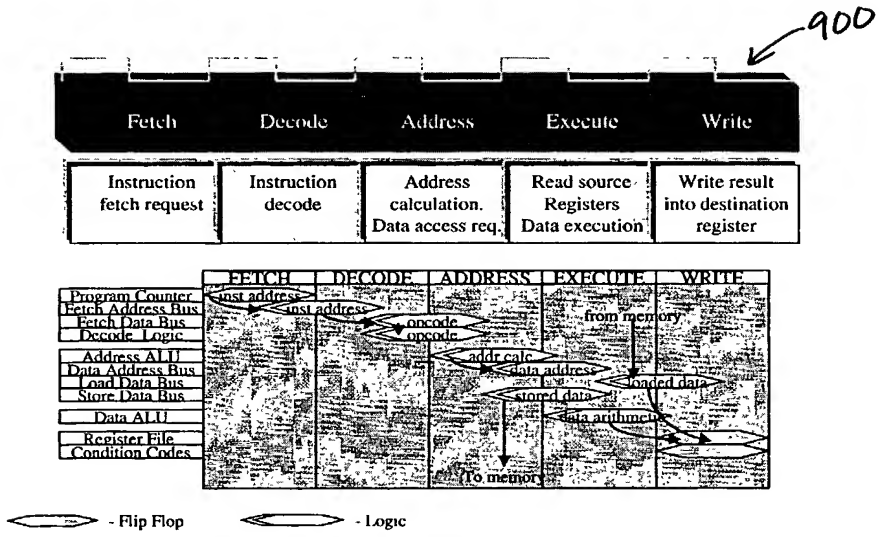

900
Fetch
Decode
Address
Execute
Write
Instruction fetch request
Instruction decode
Address calculation. Data access req.
Read source Registers Data execution
Write result into destination register
FETCH
DECODE
ADDRESS
EXECUTE
WRITE
Program Counter
Fetch Address Bus
Fetch Data Bus
Decode Logic
Address ALU
Data Address Bus
Load Data Bus
Store Data Bus
Data ALU
Register File
Condition Codes
inst address
inst address
opcode
opcode
addr calc
data address
stored data
from memory
loaded data
data arithmetic
To memory
- Flip Flop
- Logic

Fig. 73

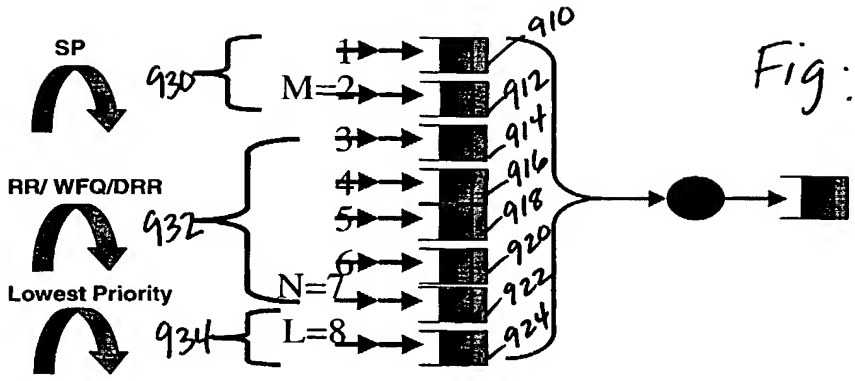

SP
930
RR/ WFQ/DRR
932
Lowest Priority
934
M=2
N=7
L=8
910
912
914
916
918
920
922
924

Fig: 74

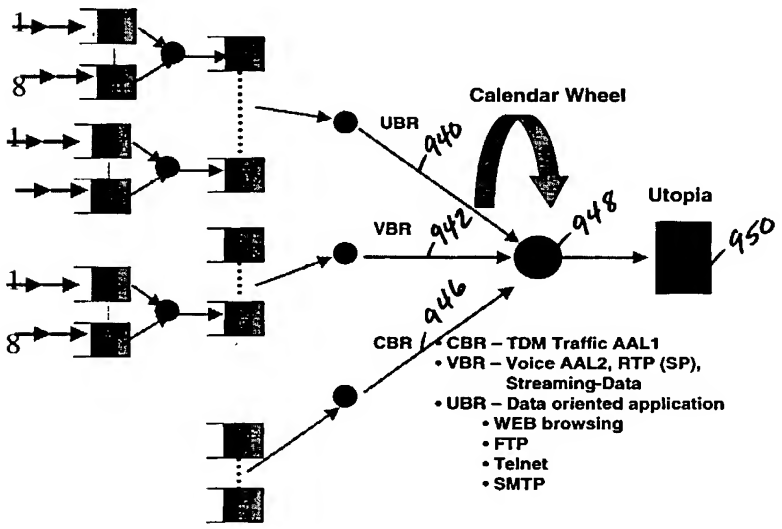

Calendar Wheel
UBR
940
VBR
942
946
948
Utopia
950
CBR
• CBR – TDM Traffic AAL1
• VBR – Voice AAL2, RTP (SP), Streaming-Data
• UBR – Data oriented application
• WEB browsing
• FTP
• Telnet
• SMTP

Fig 75

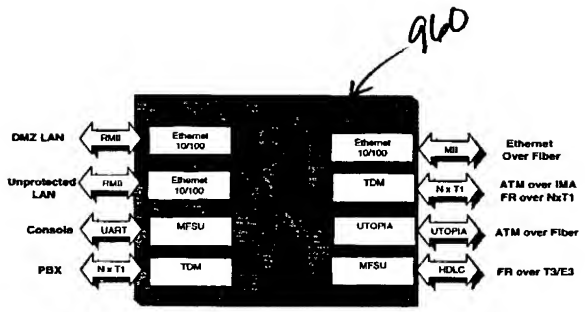

960
DMZ LAN
RMII
Ethernet 10/100
Unprotected LAN
RMII
Ethernet 10/100
Console
UART
MFSU
PBX
N x T1
TDM
Ethernet 10/100
MII
Ethernet Over Fiber
TDM
N x T1
ATM over IMA FR over NxT1
UTOPIA
UTOPIA
ATM over Fiber
MFSU
HDLC
FR over T3/E3

Fig. 76

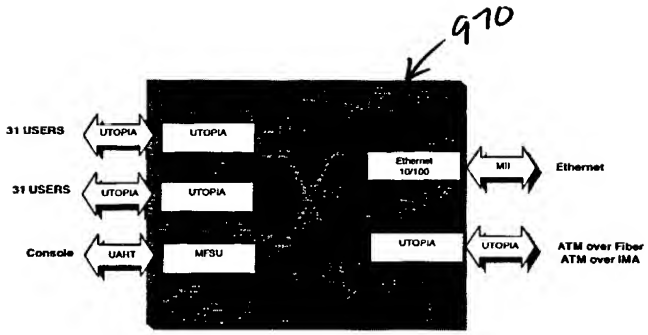

970
31 USERS
UTOPIA
UTOPIA
31 USERS
UTOPIA
UTOPIA
Console
UART
MFSU
Ethernet 10/100
MII
Ethernet
UTOPIA
UTOPIA
ATM over Fiber
ATM over IMA

Fig. 77

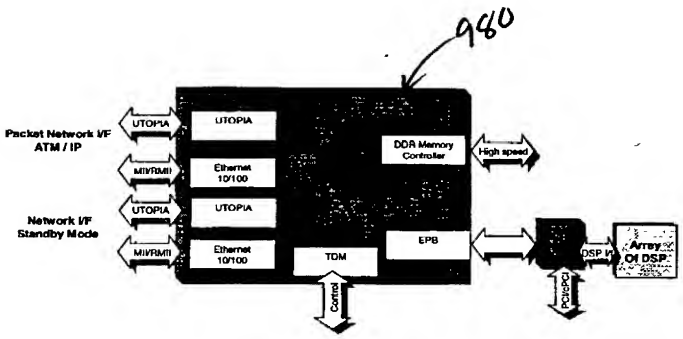

980
Packet Network I/F
ATM / IP
UTOPIA
MII/RMII
Network I/F
Standby Mode
UTOPIA
MII/RMII
UTOPIA
Ethernet
10/100
UTOPIA
Ethernet
10/100
TDM
Control
DDR Memory
Controller
High speed
EPB
DSP I/F
Array
Of DSP
PCI/PCI

Fig. 78

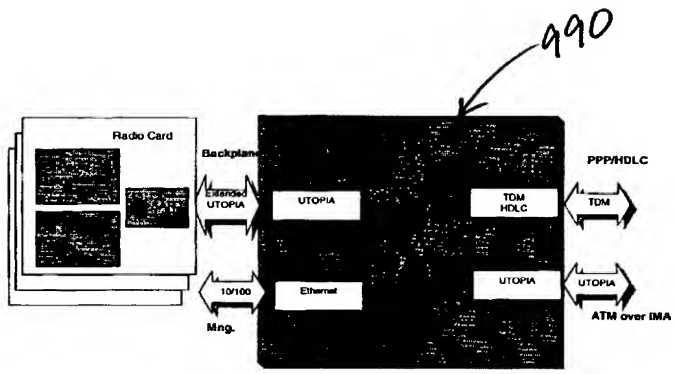

990
Radio Card
Backplane
Extended
UTOPIA
10/100
Mng.
UTOPIA
Ethernet
TDM
HDLC
UTOPIA
PPP/HDLC
TDM
UTOPIA
ATM over IMA

Fig. 79

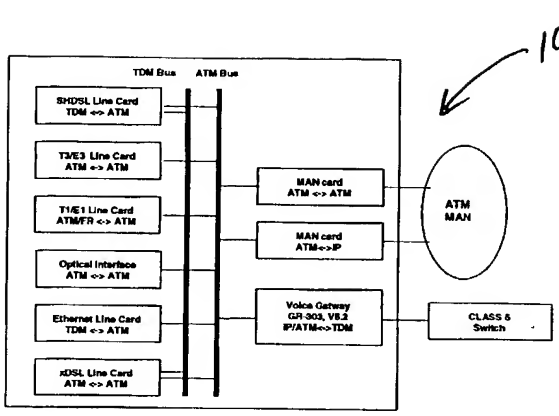

1000
TDM Bus
ATM Bus
SHDSL Line Card
TDM <-> ATM
T3/E3 Line Card
ATM <-> ATM
T1/E1 Line Card
ATM/FR <-> ATM
Optical Interface
ATM <-> ATM
Ethernet Line Card
TDM <-> ATM
xDSL Line Card
ATM <-> ATM
MAN card
ATM <-> ATM
MAN card
ATM<->IP
Voice Gatway
GR-303, V5.2
IP/ATM<->TDM
ATM
MAN
CLASS 5
Switch

Fig. 80

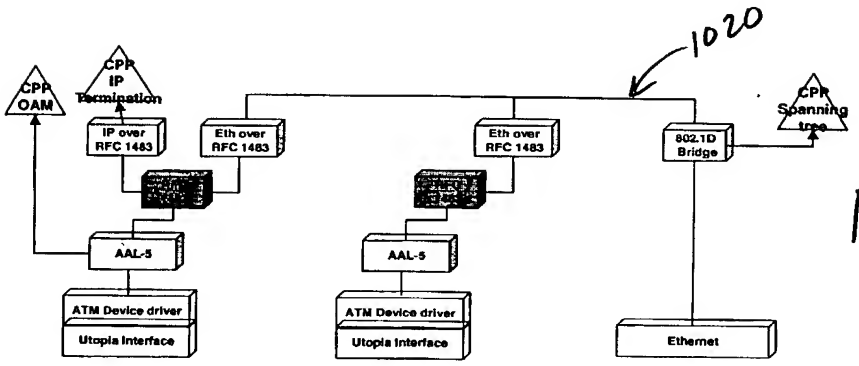

1020
CPP
OAM
CPP
IP
Termination
IP over
RFC 1483
Eth over
RFC 1483
Eth over
RFC 1483
802.1D
Bridge
CPP
Spanning
tree
AAL-5
AAL-5
ATM Device driver
Utopia Interface
ATM Device driver
Utopia Interface
Ethernet

Fig. 81

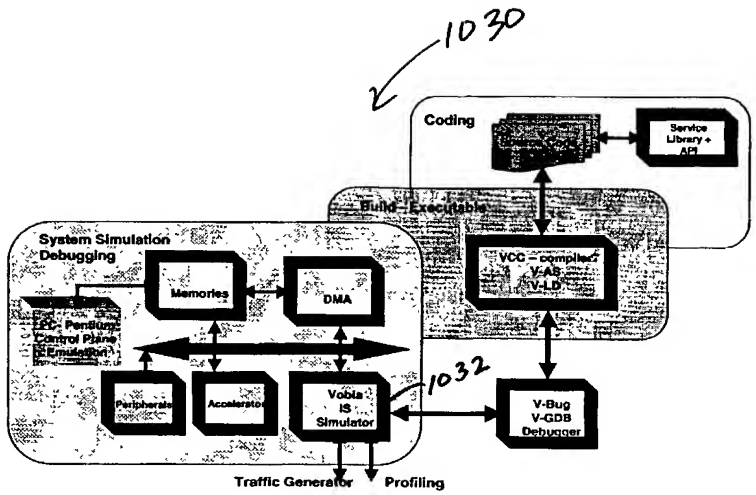

1030
Coding
Service Library + API
Build Executable
System Simulation Debugging
Memories
DMA
VCC - compiler V-AS V-LD
Peripherals
Accelerator
Vobla IS Simulator
1032
V-Bug V-GDB Debugger
Traffic Generator
Profiling

Fig. 82

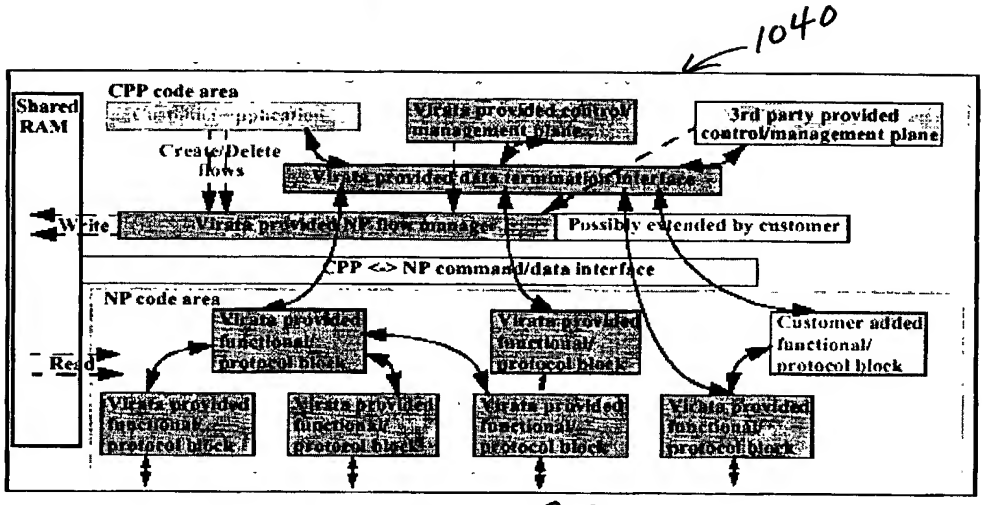

1040
Shared RAM
CPP code area
Create/Delete flows
Write
Read
3rd party provided control/management plane
Possibly extended by customer
CPP <-> NP command/data interface
NP code area
Customer added functional/ protocol block

Fig. 83

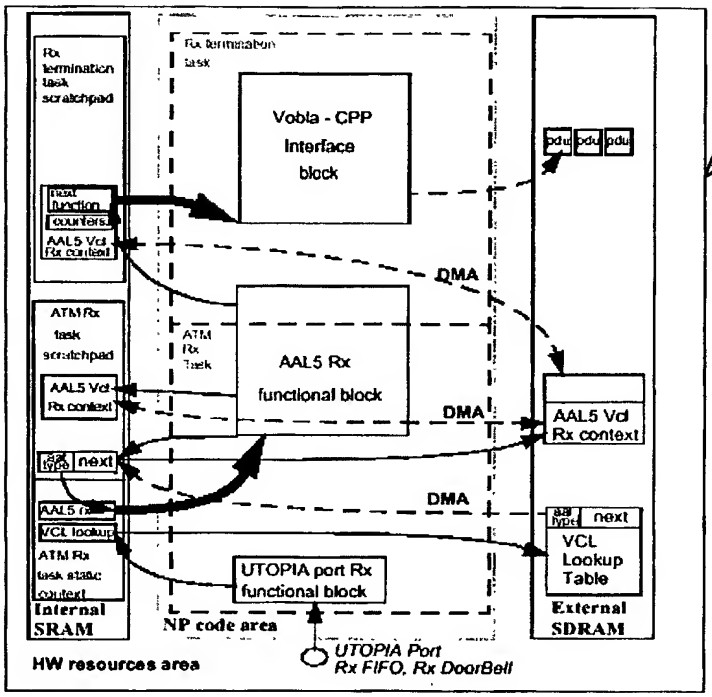

Rx termination task scratchpad
next function
counters
AAL5 Vcl Rx context
ATM Rx task scratchpad
AAL5 Vcl Rx context
type
next
AAL5 n
VCL lookup
ATM Rx task static context
Internal SRAM
HW resources area
Rx termination task
Vobla - CPP Interface block
ATM Rx Task
AAL5 Rx functional block
UTOPIA port Rx functional block
NP code area
UTOPIA Port
Rx FIFO, Rx DoorBell
pdu
pdu
pdu
DMA
DMA
DMA
AAL5 Vcl Rx context
type
next
VCL Lookup Table
External SDRAM
1050

Fig. 84

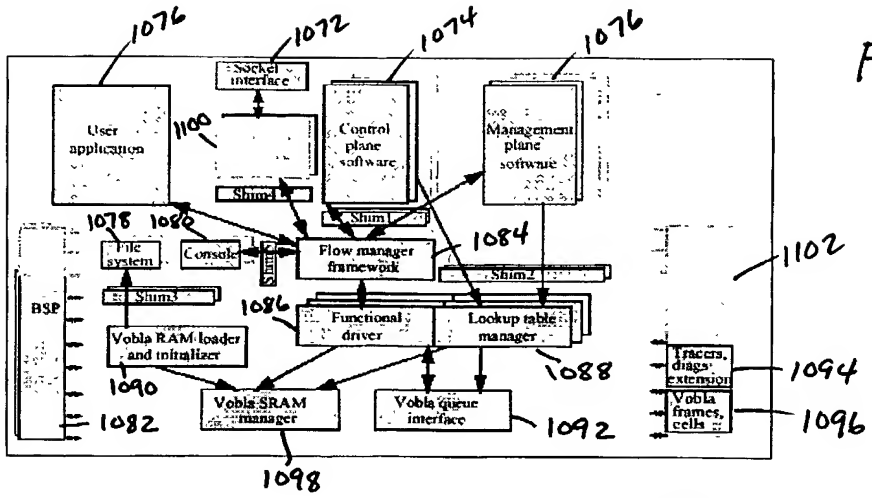

1076
1072
1074
1076
Socket interface
User application
1100
Control plane software
Management plane software
Shim4
Shim
1078
1080
File system
Console
Shim1
Flow manager framework
1084
1102
Shim2
BSP
Shim3
1086
Functional driver
Lookup table manager
Vobla RAM loader and initializer
1088
Tracers, diags, extension
1094
1090
Vobla SRAM manager
Vobla queue interface
Vobla frames, cells
1096
1082
1092
1098

Fig. 85

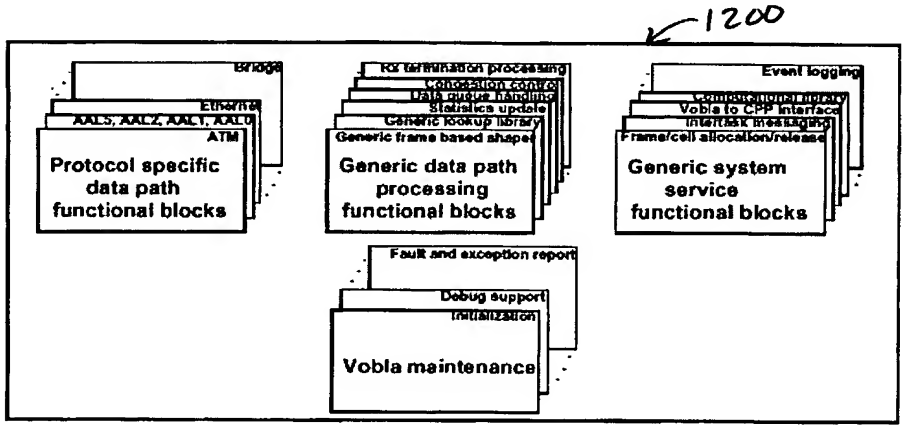

1200
Bridge
Ethernet
AAL5, AAL2, AAL1, AAL0
ATM
Protocol specific data path functional blocks
Rx termination processing
Connection control
Data queue handling
Statistics update
Generic lookup library
Generic frame based shaper
Generic data path processing functional blocks
Event logging
Computational library
Vobla to CPP interface
InterTask messaging
Frame/cell allocation/release
Generic system service functional blocks
Fault and exception report
Debug support
Initialization
Vobla maintenance

Fig. 86

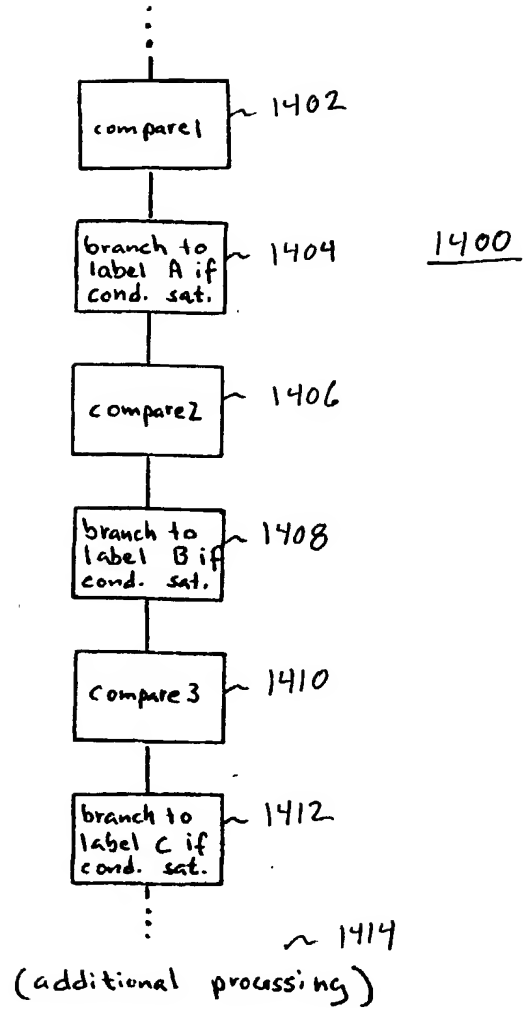


FIG. 87
(PRIOR ART)

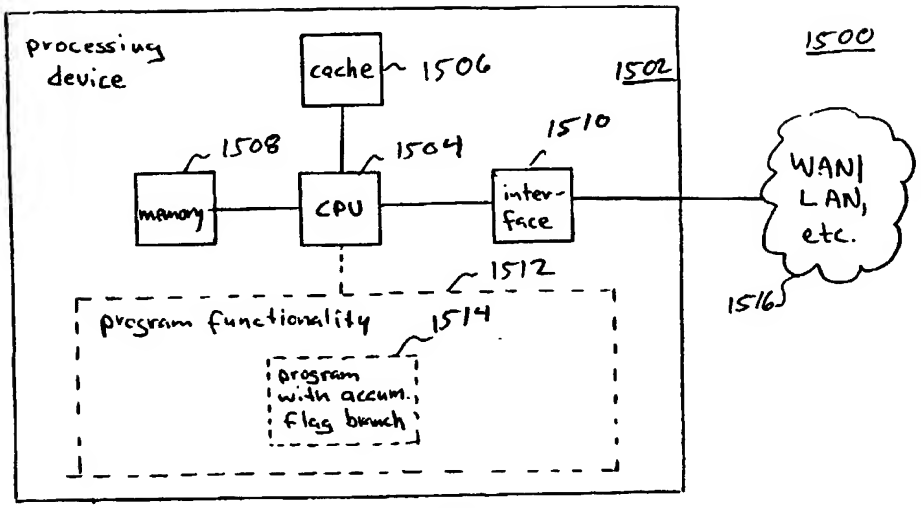

processing device
cache
1506
1502
1500
1508
1504
1510
memory
CPU
inter-face
WAN, LAN, etc.
1512
program functionality
1514
program with accum. flag branch
1516

FIG. 88

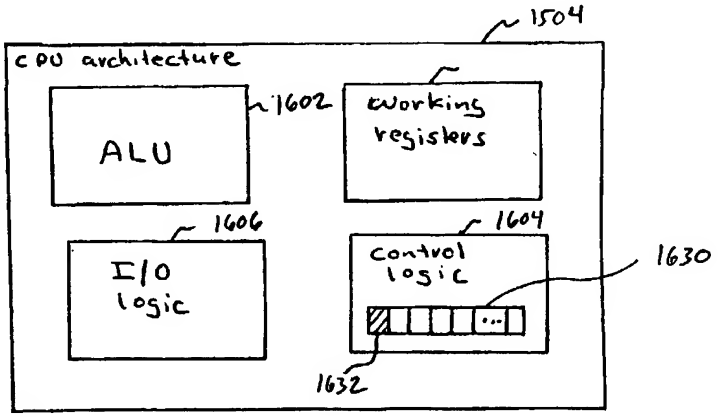

1504
CPU architecture
ALU
1602
working registers
1606
1604
I/O logic
control logic
1630
1632

FIG. 89

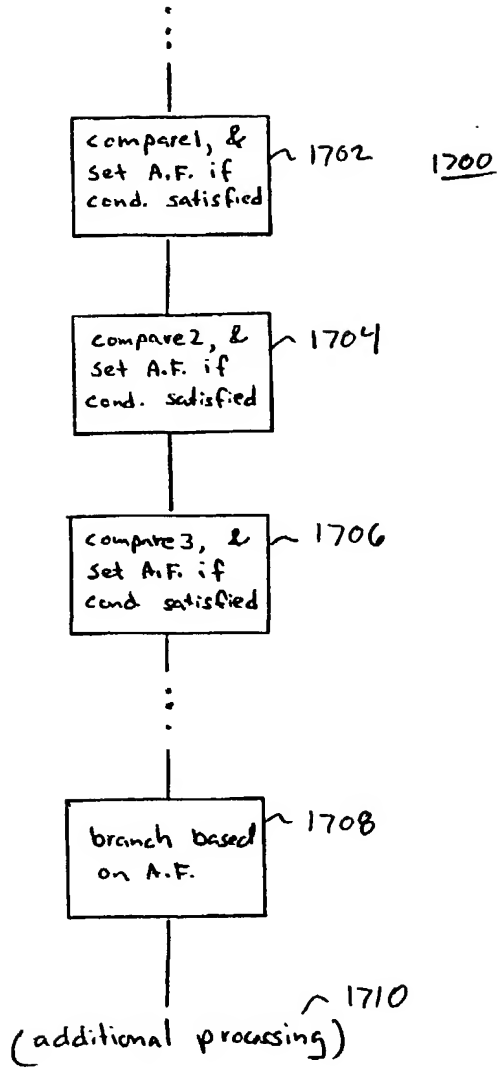

compare1, & set A.F. if cond. satisfied
1702
1700
compare2, & set A.F. if cond. satisfied
1704
compare3, & set A.F. if cond satisfied
1706
branch based on A.F.
1708
1710
(additional processing)

FIG. 90